The Cook-Levin theorem

Frank J. Balbach

March 17, 2025

Abstract

The Cook-Levin theorem states that deciding the satisfiability of Boolean formulas in conjunctive normal form is $\mathcal{NP}$-complete. This entry formalizes a proof of this theorem based on the textbook *Computational Complexity: A Modern Approach* by Arora and Barak. It contains definitions of deterministic multi-tape Turing machines, the complexity classes $\mathcal{P}$ and $\mathcal{NP}$, polynomial-time many-one reduction, and the decision problem `SAT`. For the $\mathcal{NP}$-hardness of `SAT`, the proof first shows that every polynomial-time computation can be performed by a two-tape oblivious Turing machine. An $\mathcal{NP}$ problem can then be reduced to `SAT` by a polynomial-time Turing machine that encodes computations of the problem's oblivious two-tape verifier Turing machine as formulas in conjunctive normal form.

Contents

Chapter 1

Introduction

The Cook-Levin theorem states that the problem `SAT` of deciding the satisfiability of Boolean formulas in conjunctive normal form is $\mathcal{NP}$-complete [4, 10]. This article formalizes a proof of this theorem based on the textbook *Computational Complexity: A Modern Approach* by Arora and Barak [2].

1.1 Outline

We start out in Chapter 2 with a definition of multi-tape Turing machines (TMs) slightly modified from Arora and Barak's definition. The remainder of the chapter is devoted to constructing ever more complex machines for arithmetic on binary numbers, evaluating polynomials, and performing basic operations on lists of numbers and even lists of lists of numbers.

Specifying Turing machines and proving their correctness and running time is laborious at the best of times. We slightly alleviate the seemingly inevitable tedium of this by defining elementary reusable Turing machines and introducing ways of composing them sequentially as well as in if-then-else branches and while loops. Together with the representation of natural numbers and lists, we thus get something faintly resembling a structured programming language of sorts.

In Chapter 3 we introduce some basic concepts of complexity theory, such as $\mathcal{P}$, $\mathcal{NP}$, and polynomial-time many-one reduction. Following Arora and Barak the complexity class $\mathcal{NP}$ is defined via verifier Turing machines rather than nondeterministic machines, and so the deterministic TMs introduced in the previous chapter suffice for all definitions. To flesh out the chapter a little we formalize obvious proofs of $\mathcal{P} \subseteq \mathcal{NP}$ and the transitivity of the reducibility relation, although neither result is needed for proving the Cook-Levin theorem.

Chapter 4 introduces the problem `SAT` as a language over bit strings. Boolean formulas in conjunctive normal form (CNF) are represented as lists of clauses, each consisting of a list of literals encoded in binary numbers. The list of lists of numbers "data type" defined in Chapter 2 will come in handy at this point.

The proof of the Cook-Levin theorem has two parts: Showing that `SAT` is in $\mathcal{NP}$ and showing that `SAT` is $\mathcal{NP}$-hard, that is, that every language in $\mathcal{NP}$ can be reduced to `SAT` in polynomial time. The first part, also proved in Chapter 4, is fairly easy: For a satisfiable CNF formula, a satisfying assignment can be given in roughly the size of the formula, because only the variables in the formula need be assigned a truth value. Moreover whether an assignment satisfies a CNF formula can be verified easily.

The hard part is showing the $\mathcal{NP}$-hardness of `SAT`. The first step (Chapter 5) is to show that every polynomial-time computation on a multi-tape TM can be performed in polynomial time on a two-tape *oblivious* TM. Oblivious means that the sequence of positions of the Turing machine's tape heads depends only on the *length* of the input. Thus any language in $\mathcal{NP}$ has a polynomial-time two-tape oblivious verifier TM. In Chapter 6 the proof goes on to show how the computations of such a machine can be mapped to CNF formulas such that a CNF formula is satisfiable if and only if the underlying computation was for a string in the language `SAT` paired with a certificate. Finally in Chapter 7 and Chapter 8 we construct a Turing machine that carries out the reduction in polynomial time.

1.2 Related work

The Cook-Levin theorem has been formalized before. Gamboa and Cowles [8] present a formalization in ACL2 [3]. They formalize $\mathcal{NP}$ and reducibility in terms of Turing machines, but analyze the running time of the reduction from $\mathcal{NP}$-languages to `SAT` in a different, somewhat ad-hoc, model of computation that they call "the major weakness" of their formalization.

Employing Coq [13], Gäher and Kunze [7] define $\mathcal{NP}$ and reducibility in the computational model "call-by-value λ-calculus L" introduced by Forster and Smolka [6]. They show the $\mathcal{NP}$-completeness of `SAT` in this framework. Turing machines appear in an intermediate problem in the chain of reductions from $\mathcal{NP}$ languages to `SAT`, but are not used to show the polynomiality of the reduction. Nevertheless, this is likely the first formalization of the Cook-Levin theorem where both the complexity theoretic concepts and the proof of the polynomiality of the reduction use the same model of computation.

With regards to Isabelle, Xu et al. [15] provide a formalization of single-tape Turing machines with a fixed binary alphabet in the computability theory setting and construct a universal TM. While I was putting the finishing touches on this article, Dalvit and Thiemann [5] published a formalization of (deterministic and nondeterministic) multi-tape and single-tape Turing machines and showed how to simulate the former on the latter with quadratic slowdown. Moreover, Thiemann and Schmidinger [14] prove the $\mathcal{NP}$-completeness of the Multiset Ordering problem, without, however, proving the polynomial-time computability of the reduction.

This article uses Turing machines as model of computation for both the complexity theoretic concepts and the running time analysis of the reduction. It is thus most similar to Gäher and Kunze's work, but has a more elementary, if not brute-force, flavor to it.

1.3 The core concepts

The proof of the Cook-Levin theorem awaits us in Section 8.4 on the very last page of this article. The way there is filled with definitions of Turing machines, correctness proofs for Turing machines, and running time-bound proofs for Turing machines, all of which can easily drown out the more relevant concepts. For instance, for verifying that the theorem on the last page really is the Cook-Levin theorem, only a small fraction of this article is relevant, namely the definitions of $\mathcal{NP}$-completeness and of `SAT`. Recursively breaking down these definitions yields:

- $\mathcal{NP}$-completeness: Section 3.1
  - languages: Section 3.1
  - $\mathcal{NP}$-hard: Section 3.1
    - $\mathcal{NP}$: Section 3.1
      - Turing machines: Section 2.1.1
      - computing a function: Section 2.1.2
      - pairing strings: Section 2.1.3
      - Big-Oh, polynomial: Section 2.1.4
    - polynomial-time many-one reduction: Section 3.1
- `SAT`: Section 4.1.3
  - literal, clause, CNF formula, assignment, satisfiability: Section 4.1.1
  - representing CNF formulas as strings: Section 4.1.3
    - string: Section 2.1.1
    - CNF formula: Section 4.1.1
    - mapping between symbols and strings: Section 2.1.2
    - mapping between binary and quaternary alphabets: Section 2.10.1
    - lists of lists of natural numbers: Section 2.9.1
      - binary representation of natural numbers: Section 2.7.1
      - lists of natural numbers: Section 2.8.1

In other words the Sections 2.1, 2.7.1, 2.8.1, 2.9.1, 2.10.1, 3.1, 4.1.1, and 4.1.3 cover all definitions for formalizing the statement "`SAT` is $\mathcal{NP}$-complete".

Chapter 2

Turing machines

This chapter introduces Turing machines as a model of computing functions within a running-time bound. Despite being quite intuitive, Turing machines are notoriously tedious to work with. And so most of the rest of the chapter is devoted to making this a little easier by providing means of combining TMs and a library of reusable TMs for common tasks.

The basic idea (Sections 2.1 and 2.2) is to treat Turing machines as a kind of GOTO programming language. A state of a TM corresponds to a line of code executing a rather complex command that, depending on the symbols read, can write symbols, move tape heads, and jump to another state (that is, line of code). States are identified by line numbers. This makes it easy to execute TMs in sequence by concatenating two TM "programs". On top of the GOTO implicit in all commands, we then define IF and WHILE in the traditional way (Section 2.3). This makes TMs more composable.

The interpretation of states as line numbers deprives TMs of the ability to memorize values "in states", for example, the carry bit during a binary addition. In Section 2.5 we recover some of this flexibility.

Being able to combine TMs is helpful, but we also need TMs to combine. This takes up most of the remainder of the chapter. We start with simple operations, such as moving a tape head to the next blank symbol or copying symbols between tapes (Section 2.4). Extending our programming language analogy for more complex TMs, we identify tapes with variables, so that a tape contains a value of a specific type, such as a number or a list of numbers. In the remaining Sections 2.7 to 2.12 we define these "data types" and devise TMs for operations over them.

It would be an exaggeration to say all this makes working with Turing machines easy or fun. But at least it makes TMs somewhat more feasible to use for complexity theory, as witnessed by the subsequent chapters.

2.1 Basic definitions

theory *Basics*
  imports *Main*
begin

While Turing machines are fairly simple, there are still a few parts to define, especially if one allows multiple tapes and an arbitrary alphabet: states, tapes (read-only or read-write), cells, tape heads, head movements, symbols, and configurations. Beyond these are more semantic aspects like executing one or many steps of a Turing machine, its running time, and what it means for a TM to "compute a function". Our approach at formalizing all this must look rather crude compared to Dalvit and Thiemann's [5], but still it does get the job done.

For lack of a better place, this section also introduces a minimal version of Big-Oh, polynomials, and a pairing function for strings.

2.1.1 Multi-tape Turing machines

Arora and Barak [2, p. 11] define multi-tape Turing machines with these features:

- There are $k \geq 2$ infinite one-directional tapes, and each has one head.

- The first tape is the input tape and read-only; the other $k-1$ tapes can be written to.
- The tape alphabet is a finite set Γ containing at least the blank symbol $\Box$, the start symbol $\triangleright$, and the symbols $\mathbf{0}$ and $\mathbf{1}$.
- There is a finite set Q of states with start state and halting state $q_{start}, q_{halt} \in Q$.
- The behavior is described by a transition function $\delta\colon\ Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$. If the TM is in a state q and the symbols $g_1, \dots, g_k$ are under the k tape heads and $\delta(q, (g_1, \dots, g_k)) = (q', (g'_2, \dots, g'_k), (d_1, \dots, d_k))$, then the TM writes $g'_2, \dots, g'_k$ to the writable tapes, moves the tape heads in the direction (Left, Stay, or Right) indicated by the $d_1, \dots, d_k$ and switches to state q'.

Syntax

An obvious data type for the direction a tape head can move:

datatype *direction* = *Left* | *Stay* | *Right*

We simplify the definition a bit in that we identify both symbols and states with natural numbers:

- We set $\Gamma = \{0, 1, \dots, G-1\}$ for some $G \geq 4$ and represent the symbols $\Box$, $\triangleright$, $\mathbf{0}$, and $\mathbf{1}$ by the numbers 0, 1, 2, and 3, respectively. We represent an alphabet Γ by its size G.
- We let the set of states be of the form $\{0, 1, \dots, Q\}$ for some $Q \in \mathbb{N}$ and set the start state $q_{start} = 0$ and halting state $q_{halt} = Q$.

The last item presents a fundamental difference to the textbook definition, because it requires that Turing machines with $q_{start} = q_{halt}$ have exactly one state, whereas the textbook definition allows them arbitrarily many states. However, if $q_{start} = q_{halt}$ then the TM starts in the halting state and thus does not actually do anything. But then it does not matter if there are other states besides that one start/halting state. Our simplified definition therefore does not restrict the expressive power of TMs. It does, however, simplify composing them.

The type *nat* is used for symbols and for states.

type-synonym *state* = *nat*

type-synonym *symbol* = *nat*

It is confusing to have the numbers 2 and 3 represent the symbols $\mathbf{0}$ and $\mathbf{1}$. The next abbreviations try to hide this somewhat. The glyphs for symbols number 4 and 5 are chosen arbitrarily. While we will encounter Turing machines with huge alphabets, only the following symbols will be used literally:

abbreviation (*input*) *blank-symbol* :: *nat* (‹$\Box$›) **where** $\Box \equiv$ *0*
abbreviation (*input*) *start-symbol* :: *nat* (‹$\triangleright$›) **where** $\triangleright \equiv$ *1*
abbreviation (*input*) *zero-symbol* :: *nat* (‹$\mathbf{0}$›) **where** $\mathbf{0} \equiv$ *2*
abbreviation (*input*) *one-symbol* :: *nat* (‹$\mathbf{1}$›) **where** $\mathbf{1} \equiv$ *3*
abbreviation (*input*) *bar-symbol* :: *nat* (‹$|$›) **where** $| \equiv$ *4*
abbreviation (*input*) *sharp-symbol* :: *nat* (‹$\sharp$›) **where** $\sharp \equiv$ *5*

unbundle *no abs-syntax*

Tapes are infinite in one direction, so each cell can be addressed by a natural number. Likewise the position of a tape head is a natural number. The contents of a tape are represented by a mapping from cell numbers to symbols. A *tape* is a pair of tape contents and head position:

type-synonym *tape* = (*nat* $\Rightarrow$ *symbol*) $\times$ *nat*

Our formalization of Turing machines begins with a data type representing a more general concept, which we call *machine*, and later adds a predicate to define which machines are *Turing* machines. In this generalization the number k of tapes is arbitrary, although machines with zero tapes are of little interest. Also, all tapes are writable and the alphabet is not limited, that is, $\Gamma = \mathbb{N}$. The transition function becomes $\delta\colon\ \{0, \dots, Q\} \times \mathbb{N}^k \to \{0, \dots, Q\} \times \mathbb{N}^k \times \{L, S, R\}^k$ or, saving us one occurrence of k, $\delta\colon\ \{0, \dots, Q\} \times \mathbb{N}^k \to \{0, \dots, Q\} \times (\mathbb{N} \times \{L, S, R\})^k$.

The transition function δ has a fixed behavior in the state $q_{halt} = Q$ (namely making the machine do nothing). Hence δ needs to be specified only for the Q states $0, \dots, Q-1$ and thus can be given as a sequence $\delta_0, \dots, \delta_{Q-1}$ where each δ_q is a function

$$\delta_q \colon \mathbb{N}^k \to \{0, \dots, Q\} \times (\mathbb{N} \times \{L, S, R\})^k. \tag{2.1}$$

Going one step further we allow the machine to jump to any state in $\mathbb{N}$, and we will treat any state $q \geq Q$ as a halting state. The δ_q are then

$$\delta_q \colon \mathbb{N}^k \to \mathbb{N} \times (\mathbb{N} \times \{L, S, R\})^k. \tag{2.2}$$

Finally we allow inputs and outputs of arbitrary length, turning the δ_q into

$$\delta_q \colon \mathbb{N}^* \to \mathbb{N} \times (\mathbb{N} \times \{L, S, R\})^*.$$

Such a δ_q will be called a *command*, and the elements of $\mathbb{N} \times \{L, S, R\}$ will be called *actions*. An action consists of writing a symbol to a tape at the current tape head position and then moving the tape head.

type-synonym *action* = *symbol* × *direction*

A command maps the list of symbols read from the tapes to a follow-up state and a list of actions. It represents the machine's behavior in one state.

type-synonym *command* = *symbol list* ⇒ *state* × *action list*

Machines are then simply lists of commands. The q-th element of the list represents the machine's behavior in state q. The halting state of a machine M is *length M*, but there is obviously no such element in the list.

type-synonym *machine* = *command list*

Commands in this general form are too amorphous. We call a command *well-formed* for k tapes and the state space Q if on reading k symbols it performs k actions and jumps to a state in $\{0, \dots, Q\}$. A well-formed command corresponds to (2.1).

definition *wf-command* :: *nat* ⇒ *nat* ⇒ *command* ⇒ *bool* **where**
  wf-command k Q cmd ≡ ∀ *gs*. *length gs* = *k* ⟶ *length* (*snd* (*cmd gs*)) = *k* ∧ *fst* (*cmd gs*) ≤ *Q*

A well-formed command is a *Turing command* for k tapes and alphabet G if it writes only symbols from G when reading symbols from G and does not write to tape 0; that is, it writes to tape 0 the symbol it read from tape 0.

definition *turing-command* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *command* ⇒ *bool* **where**
  turing-command k Q G cmd ≡
    wf-command k Q cmd ∧
    (∀ *gs*. *length gs* = *k* ⟶
      ((∀ *i*<*k*. *gs* ! *i* < *G*) ⟶ (∀ *i*<*k*. *fst* (*snd* (*cmd gs*) ! *i*) < *G*)) ∧
       (*k* > *0* ⟶ *fst* (*snd* (*cmd gs*) ! *0*) = *gs* ! *0*))

A *Turing machine* is a machine with at least two tapes and four symbols and only Turing commands.

definition *turing-machine* :: *nat* ⇒ *nat* ⇒ *machine* ⇒ *bool* **where**
  turing-machine k G M ≡ *k* ≥ *2* ∧ *G* ≥ *4* ∧ (∀ *cmd*∈*set M*. *turing-command k* (*length M*) *G cmd*)

Semantics

Next we define the semantics of machines. The state and the list of tapes make up the *configuration* of a machine. The semantics are given as functions mapping configurations to follow-up configurations.

type-synonym *config* = *state* × *tape list*

We start with the semantics of a single command. An action affects a tape in the following way. For the head movements we imagine the tapes having cell 0 at the left and the cell indices growing rightward.

fun *act* :: *action* ⇒ *tape* ⇒ *tape* **where**
  act (*w*, *m*) *tp* =
    ((*fst tp*)(*snd tp*:=*w*),
     case m of Left ⇒ *snd tp* − *1* | *Stay* ⇒ *snd tp* | *Right* ⇒ *snd tp* + *1*)

Reading symbols from one tape, from all tapes, and from configurations:

abbreviation *tape-read* :: *tape* ⇒ *symbol* (‹|.|›) **where**
  |.| *tp* ≡ *fst tp* (*snd tp*)

definition *read* :: *tape list* ⇒ *symbol list* **where**
  read tps ≡ *map tape-read tps*

abbreviation *config-read* :: *config* ⇒ *symbol list* **where**
  config-read cfg ≡ *read* (*snd cfg*)

The semantics of a command:

definition *sem* :: *command* ⇒ *config* ⇒ *config* **where**
  sem cmd cfg ≡
    let (*newstate*, *actions*) = *cmd* (*config-read cfg*)
    in (*newstate*, *map* (λ(*a*, *tp*). *act a tp*) (*zip actions* (*snd cfg*)))

The semantics of one step of a machine consist in the semantics of the command corresponding to the state the machine is in. The following definition ensures that the configuration does not change when it is in a halting state.

definition *exe* :: *machine* ⇒ *config* ⇒ *config* **where**
  exe M cfg ≡ *if fst cfg* < *length M then sem* (*M* ! (*fst cfg*)) *cfg else cfg*

Executing a machine M for multiple steps:

fun *execute* :: *machine* ⇒ *config* ⇒ *nat* ⇒ *config* **where**
  execute M cfg 0 = *cfg* |
  execute M cfg (*Suc t*) = *exe M* (*execute M cfg t*)

We have defined the semantics for arbitrary machines, but most lemmas we are going to prove about *exe*, *execute*, etc. will require the commands to be somewhat well-behaved, more precisely to map lists of k symbols to lists of k actions, as shown in (2.2). We will call such commands *proper*.

abbreviation *proper-command* :: *nat* ⇒ *command* ⇒ *bool* **where**
  proper-command k cmd ≡ ∀ *gs*. *length gs* = *k* ⟶ *length* (*snd* (*cmd gs*)) = *length gs*

Being proper is a weaker condition than being well-formed. Since *exe* treats the state Q and the states $q > Q$ the same, we do not need the Q-closure property of well-formedness for most lemmas about semantics.

Next we introduce a number of abbreviations for components of a machine and aspects of its behavior. In general, symbols between bars $|\cdot|$ represent operations on tapes, inside angle brackets $< \cdot >$ operations on configurations, between colons $: \cdot :$ operations on lists of tapes, and inside brackets $[\cdot]$ operations on state/action-list pairs. As for the symbol inside the delimiters, a dot (.) refers to a tape symbol, a colon (:) to the entire tape contents, and a hash (#) to a head position; an equals sign (=) means some component of the left-hand side is changed. An exclamation mark (!) accesses an element in a list on the left-hand side term.

abbreviation *config-length* :: *config* ⇒ *nat* (‹||-||›) **where**
  config-length cfg ≡ *length* (*snd cfg*)

abbreviation *tape-move-right* :: *tape* ⇒ *nat* ⇒ *tape* (**infixl** ‹|+|› *60*) **where**
  tp |+| *n* ≡ (*fst tp*, *snd tp* + *n*)

abbreviation *tape-move-left* :: *tape* ⇒ *nat* ⇒ *tape* (**infixl** ‹|−|› *60*) **where**
  tp |−| *n* ≡ (*fst tp*, *snd tp* − *n*)

abbreviation *tape-move-to* :: *tape* ⇒ *nat* ⇒ *tape* (**infixl** ‹|#=|› *60*) **where**
  tp |#=| *n* ≡ (*fst tp*, *n*)

abbreviation *tape-write* :: *tape* ⇒ *symbol* ⇒ *tape* (**infixl** ‹|:=|› *60*) **where**
  tp |:=| *h* ≡ ((*fst tp*) (*snd tp* := *h*), *snd tp*)

abbreviation *config-tape-by-no* :: *config* ⇒ *nat* ⇒ *tape* (**infix** ‹<!>› *90*) **where**

cfg <!> j ≡ snd cfg ! j

abbreviation *config-contents-by-no* :: *config* ⇒ *nat* ⇒ (*nat* ⇒ *symbol*) (**infix** ‹<:>› *100*) **where**
cfg <:> *j* ≡ *fst* (*cfg* <!> *j*)

abbreviation *config-pos-by-no* :: *config* ⇒ *nat* ⇒ *nat* (**infix** ‹<#>› *100*) **where**
cfg <#> *j* ≡ *snd* (*cfg* <!> *j*)

abbreviation *config-symbol-read* :: *config* ⇒ *nat* ⇒ *symbol* (**infix** ‹<.>› *100*) **where**
cfg <.> *j* ≡ (*cfg* <:> *j*) (*cfg* <#> *j*)

abbreviation *config-update-state* :: *config* ⇒ *nat* ⇒ *config* (**infix** ‹<+=>› *90*) **where**
cfg <+=> *q* ≡ (*fst cfg* + *q*, *snd cfg*)

abbreviation *tapes-contents-by-no* :: *tape list* ⇒ *nat* ⇒ (*nat* ⇒ *symbol*) (**infix** ‹:::› *100*) **where**
tps ::: *j* ≡ *fst* (*tps* ! *j*)

abbreviation *tapes-pos-by-no* :: *tape list* ⇒ *nat* ⇒ *nat* (**infix** ‹:#:› *100*) **where**
tps :#: *j* ≡ *snd* (*tps* ! *j*)

abbreviation *tapes-symbol-read* :: *tape list* ⇒ *nat* ⇒ *symbol* (**infix** ‹:.:› *100*) **where**
tps :.: *j* ≡ (*tps* ::: *j*) (*tps* :#: *j*)

abbreviation *jump-by-no* :: *state* × *action list* ⇒ *state* (‹[∗] -› [*90*]) **where**
[∗] *sas* ≡ *fst sas*

abbreviation *actions-of-cmd* :: *state* × *action list* ⇒ *action list* (‹[!!] -› [*100*] *100*) **where**
[!!] *sas* ≡ *snd sas*

abbreviation *action-by-no* :: *state* × *action list* ⇒ *nat* ⇒ *action* (**infix** ‹[!]› *90*) **where**
sas [!] *j* ≡ *snd sas* ! *j*

abbreviation *write-by-no* :: *state* × *action list* ⇒ *nat* ⇒ *symbol* (**infix** ‹[.]› *90*) **where**
sas [.] *j* ≡ *fst* (*sas* [!] *j*)

abbreviation *direction-by-no* :: *state* × *action list* ⇒ *nat* ⇒ *direction* (**infix** ‹[~]› *100*) **where**
sas [~] *j* ≡ *snd* (*sas* [!] *j*)

Symbol sequences consisting of symbols from an alphabet *G*:

abbreviation *symbols-lt* :: *nat* ⇒ *symbol list* ⇒ *bool* **where**
symbols-lt G rs ≡ ∀ *i*<*length rs*. *rs* ! *i* < *G*

We will frequently have to show that commands are proper or Turing commands.

lemma *turing-commandI* [*intro*]:
assumes ⋀*gs*. *length gs* = *k* ⟹ *length* ([!!] *cmd gs*) = *length gs*
and ⋀*gs*. *length gs* = *k* ⟹ (⋀*i*. *i* < *length gs* ⟹ *gs* ! *i* < *G*) ⟹ (⋀*j*. *j* < *length gs* ⟹ *cmd gs* [.] *j* < *G*)
and ⋀*gs*. *length gs* = *k* ⟹ *k* > *0* ⟹ *cmd gs* [.] *0* = *gs* ! *0*
and ⋀*gs*. *length gs* = *k* ⟹ [∗] (*cmd gs*) ≤ *Q*
shows *turing-command k Q G cmd*
using *assms turing-command-def wf-command-def* **by** *simp*

lemma *turing-commandD*:
assumes *turing-command k Q G cmd* **and** *length gs* = *k*
shows *length* ([!!] *cmd gs*) = *length gs*
and (⋀*i*. *i* < *length gs* ⟹ *gs* ! *i* < *G*) ⟹ (⋀*j*. *j* < *length gs* ⟹ *cmd gs* [.] *j* < *G*)
and *k* > *0* ⟹ *cmd gs* [.] *0* = *gs* ! *0*
and ⋀*gs*. *length gs* = *k* ⟹ [∗] (*cmd gs*) ≤ *Q*
using *assms turing-command-def wf-command-def* **by** *simp-all*

lemma *turing-command-mono*:
assumes *turing-command k Q G cmd* **and** *Q* ≤ *Q'*
shows *turing-command k Q' G cmd*
using *turing-command-def wf-command-def assms* **by** *auto*

lemma *proper-command-length*:
  assumes *proper-command k cmd* **and** *length gs = k*
  shows *length* ([!!] *cmd gs*) = *length gs*
  using *assms* **by** *simp*

abbreviation *proper-machine* :: *nat* ⇒ *machine* ⇒ *bool* **where**
  proper-machine k M ≡ ∀ *i<length M*. *proper-command k* (*M* ! *i*)

lemma *prop-list-append*:
  assumes ∀ *i<length M1*. *P* (*M1* ! *i*)
    and ∀ *i<length M2*. *P* (*M2* ! *i*)
  shows ∀ *i<length* (*M1* @ *M2*). *P* ((*M1* @ *M2*) ! *i*)
  using *assms* **by** (*simp add*: *nth-append*)

The empty Turing machine [] is the one Turing machine where the start state is the halting state, that is, $q_{start} = q_{halt} = Q = 0$. It is a Turing machine for every $k \geq 2$ and $G \geq 4$:

lemma *Nil-tm*: *G* ≥ *4* ⟹ *k* ≥ *2* ⟹ *turing-machine k G* []
  using *turing-machine-def* **by** *simp*

lemma *turing-machineI* [*intro*]:
  assumes *k* ≥ *2*
    and *G* ≥ *4*
    and ⋀*i*. *i* < *length M* ⟹ *turing-command k* (*length M*) *G* (*M* ! *i*)
  shows *turing-machine k G M*
  unfolding *turing-machine-def* **using** *assms* **by** (*metis in-set-conv-nth*)

lemma *turing-machineD*:
  assumes *turing-machine k G M*
  shows *k* ≥ *2*
    and *G* ≥ *4*
    and ⋀*i*. *i* < *length M* ⟹ *turing-command k* (*length M*) *G* (*M* ! *i*)
  using *turing-machine-def assms* **by** *simp-all*

A few lemmas about *act*, *read*, and *sem*:

lemma *act*: *act a tp* =
  ((*fst tp*)(*snd tp* := *fst a*),
   case snd a of Left ⇒ *snd tp* − *1* | *Stay* ⇒ *snd tp* | *Right* ⇒ *snd tp* + *1*)
  by (*metis act.simps prod.collapse*)

lemma *act-Stay*: *j* < *length tps* ⟹ *act* (*read tps* ! *j*, *Stay*) (*tps* ! *j*) = *tps* ! *j*
  by (*simp add*: *read-def*)

lemma *act-Right*: *j* < *length tps* ⟹ *act* (*read tps* ! *j*, *Right*) (*tps* ! *j*) = *tps* ! *j* |+| *1*
  by (*simp add*: *read-def*)

lemma *act-Left*: *j* < *length tps* ⟹ *act* (*read tps* ! *j*, *Left*) (*tps* ! *j*) = *tps* ! *j* |−| *1*
  by (*simp add*: *read-def*)

lemma *act-Stay'*: *act* (*h*, *Stay*) (*tps* ! *j*) = *tps* ! *j* |:=| *h*
  by *simp*

lemma *act-Right'*: *act* (*h*, *Right*) (*tps* ! *j*) = *tps* ! *j* |:=| *h* |+| *1*
  by *simp*

lemma *act-Left'*: *act* (*h*, *Left*) (*tps* ! *j*) = *tps* ! *j* |:=| *h* |−| *1*
  by *simp*

lemma *act-pos-le-Suc*: *snd* (*act a* (*tps* ! *j*)) ≤ *Suc* (*snd* (*tps* ! *j*))
proof −
  obtain *w m* **where** *a* = (*w*, *m*)
    by *fastforce*

then show *snd* (*act a* (*tps* ! *j*)) ≤ *Suc* (*snd* (*tps* ! *j*))
using *act-Left′ act-Stay′ act-Right′* **by** (*cases m*) *simp-all*
qed

lemma *act-changes-at-most-pos*:
assumes $i \neq$ *snd tp*
shows *fst* (*act* (*h*, *mv*) *tp*) *i* = *fst tp i*
by (*simp add*: *assms*)

lemma *act-changes-at-most-pos′*:
assumes $i \neq$ *snd tp*
shows *fst* (*act a tp*) *i* = *fst tp i*
by (*simp add*: *assms act*)

lemma *read-length*: *length* (*read tps*) = *length tps*
using *read-def* **by** *simp*

lemma *tapes-at-read*: *j* < *length tps* ⟹ (*q*, *tps*) <.> *j* = *read tps* ! *j*
unfolding *read-def* **by** *simp*

lemma *tapes-at-read′*: *j* < *length tps* ⟹ *tps* :.: *j* = *read tps* ! *j*
unfolding *read-def* **by** *simp*

lemma *read-abbrev*: *j* < ||*cfg*|| ⟹ *read* (*snd cfg*) ! *j* = *cfg* <.> *j*
unfolding *read-def* **by** *simp*

lemma *sem*:
sem cmd cfg =
(*let rs* = *read* (*snd cfg*)
in (*fst* (*cmd rs*), *map* (λ(*a*, *tp*). *act a tp*) (*zip* (*snd* (*cmd rs*)) (*snd cfg*))))
using *sem-def read-def* **by** (*metis* (*no-types*, *lifting*) *case-prod-beta*)

lemma *sem′*:
sem cmd cfg =
(*fst* (*cmd* (*read* (*snd cfg*))), *map* (λ(*a*, *tp*). *act a tp*) (*zip* (*snd* (*cmd* (*read* (*snd cfg*)))) (*snd cfg*)))
using *sem-def read-def* **by** (*metis* (*no-types*, *lifting*) *case-prod-beta*)

lemma *sem′′*:
sem cmd (*q*, *tps*) =
(*fst* (*cmd* (*read tps*)), *map* (λ(*a*, *tp*). *act a tp*) (*zip* (*snd* (*cmd* (*read tps*))) *tps*))
using *sem′* **by** *simp*

lemma *sem-num-tapes-raw*: *proper-command k cmd* ⟹ *k* = ||*cfg*|| ⟹ *k* = ||*sem cmd cfg*||
using *sem-def read-length* **by** (*simp add*: *case-prod-beta*)

lemma *sem-num-tapes2*: *turing-command k Q G cmd* ⟹ *k* = ||*cfg*|| ⟹ *k* = ||*sem cmd cfg*||
using *sem-num-tapes-raw turing-commandD*(*1*) **by** *simp*

corollary *sem-num-tapes2′*: *turing-command* ||*cfg*|| *Q G cmd* ⟹ ||*cfg*|| = ||*sem cmd cfg*||
using *sem-num-tapes2* **by** *simp*

corollary *sem-num-tapes3*: *turing-command* ||*cfg*|| *Q G cmd* ⟹ ||*cfg*|| = ||*sem cmd cfg*||
by (*simp add*: *turing-commandD*(*1*) *sem-num-tapes-raw*)

lemma *sem-fst*:
assumes *cfg′* = *sem cmd cfg* **and** *rs* = *read* (*snd cfg*)
shows *fst cfg′* = *fst* (*cmd rs*)
using *sem* **by** (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *fstI*)

lemma *sem-snd*:
assumes *proper-command k cmd*
and ||*cfg*|| = *k*
and *rs* = *read* (*snd cfg*)

and *j* < *k*
shows *sem cmd cfg* <!> *j* = *act* (*snd* (*cmd rs*) ! *j*) (*snd cfg* ! *j*)
using *assms sem′ read-length* **by** *simp*

lemma *snd-semI*:
assumes *proper-command k cmd*
and *length tps* = *k*
and *length tps′* = *k*
and ⋀*j*. *j* < *k* ⟹ *act* (*cmd* (*read tps*) [!] *j*) (*tps* ! *j*) = *tps′* ! *j*
shows *snd* (*sem cmd* (*q*, *tps*)) = *snd* (*q′*, *tps′*)
using *assms sem-snd*[*OF assms*(*1*)] *sem-num-tapes-raw* **by** (*metis nth-equalityI snd-conv*)

lemma *sem-snd-tm*:
assumes *turing-machine k G M*
and *length tps* = *k*
and *rs* = *read tps*
and *j* < *k*
and *q* < *length M*
shows *sem* (*M* ! *q*) (*q*, *tps*) <!> *j* = *act* (*snd* ((*M* ! *q*) *rs*) ! *j*) (*tps* ! *j*)
using *assms sem-snd turing-machine-def turing-commandD*(*1*) **by** (*metis nth-mem snd-conv*)

lemma *semI*:
assumes *proper-command k cmd*
and *length tps* = *k*
and *length tps′* = *k*
and *fst* (*cmd* (*read tps*)) = *q′*
and ⋀*j*. *j* < *k* ⟹ *act* (*cmd* (*read tps*) [!] *j*) (*tps* ! *j*) = *tps′* ! *j*
shows *sem cmd* (*q*, *tps*) = (*q′*, *tps′*)
using *snd-semI*[*OF assms*(*1*,*2*,*3*)] *assms*(*4*,*5*) *sem-fst* **by** (*metis prod.exhaust-sel snd-conv*)

Commands ignore the state element of the configuration they are applied to.

lemma *sem-state-indep*:
assumes *snd cfg1* = *snd cfg2*
shows *sem cmd cfg1* = *sem cmd cfg2*
using *sem-def assms* **by** *simp*

A few lemmas about *exe* and *execute*:

lemma *exe-lt-length*: *fst cfg* < *length M* ⟹ *exe M cfg* = *sem* (*M* ! (*fst cfg*)) *cfg*
using *exe-def* **by** *simp*

lemma *exe-ge-length*: *fst cfg* ≥ *length M* ⟹ *exe M cfg* = *cfg*
using *exe-def* **by** *simp*

lemma *exe-num-tapes*:
assumes *turing-machine k G M* **and** *k* = ||*cfg*||
shows *k* = ||*exe M cfg*||
using *assms sem-num-tapes2 turing-machine-def exe-def* **by** (*metis nth-mem*)

lemma *exe-num-tapes-proper*:
assumes *proper-machine k M* **and** *k* = ||*cfg*||
shows *k* = ||*exe M cfg*||
using *assms sem-num-tapes-raw turing-machine-def exe-def* **by** *metis*

lemma *execute-num-tapes-proper*:
assumes *proper-machine k M* **and** *k* = ||*cfg*||
shows *k* = ||*execute M cfg t*||
using *exe-num-tapes-proper assms* **by** (*induction t*) *simp-all*

lemma *execute-num-tapes*:
assumes *turing-machine k G M* **and** *k* = ||*cfg*||
shows *k* = ||*execute M cfg t*||
using *exe-num-tapes assms* **by** (*induction t*) *simp-all*

lemma *execute-after-halting*:
  assumes *fst* (*execute M cfg0 t*) = *length M*
  shows *execute M cfg0* (*t* + *n*) = *execute M cfg0 t*
  by (*induction n*) (*simp-all add*: *assms exe-def*)

lemma *execute-after-halting′*:
  assumes *fst* (*execute M cfg0 t*) ≥ *length M*
  shows *execute M cfg0* (*t* + *n*) = *execute M cfg0 t*
  by (*induction n*) (*simp-all add*: *assms exe-ge-length*)

corollary *execute-after-halting-ge*:
  assumes *fst* (*execute M cfg0 t*) = *length M* **and** *t* ≤ *t′*
  shows *execute M cfg0 t′* = *execute M cfg0 t*
  using *execute-after-halting assms le-Suc-ex* **by** *blast*

corollary *execute-after-halting-ge′*:
  assumes *fst* (*execute M cfg0 t*) ≥ *length M* **and** *t* ≤ *t′*
  shows *execute M cfg0 t′* = *execute M cfg0 t*
  using *execute-after-halting′ assms le-Suc-ex* **by** *blast*

lemma *execute-additive*:
  assumes *execute M cfg1 t1* = *cfg2* **and** *execute M cfg2 t2* = *cfg3*
  shows *execute M cfg1* (*t1* + *t2*) = *cfg3*
  using *assms* **by** (*induction t2 arbitrary*: *cfg3*) *simp-all*

lemma *turing-machine-execute-states*:
  assumes *turing-machine k G M* **and** *fst cfg* ≤ *length M* **and** ||*cfg*|| = *k*
  shows *fst* (*execute M cfg t*) ≤ *length M*
proof (*induction t*)
  case *0*
  then show *?case*
    by (*simp add*: *assms*(*2*))
next
  case (*Suc t*)
  then show *?case*
    using *turing-command-def assms*(*1*,*3*) *exe-def execute.simps*(*2*) *execute-num-tapes sem-fst*
      turing-machine-def wf-command-def read-length
    by (*smt* (*verit*, *best*) *nth-mem*)
qed

While running times are important, usually upper bounds for them suffice. The next predicate expresses that a machine *transits* from one configuration to another one in at most a certain number of steps.

definition *transits* :: *machine* ⇒ *config* ⇒ *nat* ⇒ *config* ⇒ *bool* **where**
  transits M cfg1 t cfg2 ≡ ∃ *t′*≤*t*. *execute M cfg1 t′* = *cfg2*

lemma *transits-monotone*:
  assumes *t* ≤ *t′* **and** *transits M cfg1 t cfg2*
  shows *transits M cfg1 t′ cfg2*
  using *assms dual-order.trans transits-def* **by** *auto*

lemma *transits-additive*:
  assumes *transits M cfg1 t1 cfg2* **and** *transits M cfg2 t2 cfg3*
  shows *transits M cfg1* (*t1* + *t2*) *cfg3*
proof−
  from *assms*(*1*) **obtain** *t1′* **where** *1*: *t1′* ≤ *t1 execute M cfg1 t1′* = *cfg2*
    using *transits-def* **by** *auto*
  from *assms*(*2*) **obtain** *t2′* **where** *2*: *t2′* ≤ *t2 execute M cfg2 t2′* = *cfg3*
    using *transits-def* **by** *auto*
  then have *execute M cfg1* (*t1′* + *t2′*) = *cfg3*
    using *execute-additive 1* **by** *simp*
  moreover have *t1′* + *t2′* ≤ *t1* + *t2*
    using *1*(*1*) *2*(*1*) **by** *simp*
  ultimately show *?thesis*

using *transits-def 1(2) 2(2)* **by** *auto*
qed

lemma *transitsI*:
assumes *execute M cfg1* $t' =$ *cfg2* **and** $t' \leq t$
shows *transits M cfg1 t cfg2*
unfolding *transits-def* **using** *assms* **by** *auto*

lemma *execute-imp-transits*:
assumes *execute M cfg1* $t =$ *cfg2*
shows *transits M cfg1 t cfg2*
unfolding *transits-def* **using** *assms* **by** *auto*

In the vast majority of cases we are only interested in transitions from the start state to the halting state. One way to look at it is the machine *transforms* a list of tapes to another list of tapes within a certain number of steps.

definition *transforms* :: *machine* ⇒ *tape list* ⇒ *nat* ⇒ *tape list* ⇒ *bool* **where**
transforms M tps t tps' ≡ *transits M* (*0*, *tps*) *t* (*length M*, *tps'*)

The previous predicate will be the standard way in which we express the behavior of a (Turing) machine. Consider, for example, the empty machine:

lemma *transforms-Nil*: *transforms* [] *tps 0 tps*
using *transforms-def transits-def* **by** *simp*

lemma *transforms-monotone*:
assumes *transforms M tps t tps'* **and** $t \leq t'$
shows *transforms M tps t' tps'*
using *assms transforms-def transits-monotone* **by** *simp*

Most often the tapes will have a start symbol in the first cell followed by a finite sequence of symbols.

definition *contents* :: *symbol list* ⇒ (*nat* ⇒ *symbol*) (‹⌊-⌋›) **where**
⌊*xs*⌋ *i* ≡ *if i = 0 then* ▷ *else if i ≤ length xs then xs* ! (*i* − *1*) *else* □

lemma *contents-at-0* [*simp*]: ⌊*zs*⌋ *0* = ▷
using *contents-def* **by** *simp*

lemma *contents-inbounds* [*simp*]: *i* > *0* ⟹ *i* ≤ *length zs* ⟹ ⌊*zs*⌋ *i* = *zs* ! (*i* − *1*)
using *contents-def* **by** *simp*

lemma *contents-outofbounds* [*simp*]: *i* > *length zs* ⟹ ⌊*zs*⌋ *i* = □
using *contents-def* **by** *simp*

When Turing machines are used to compute functions, they are started in a specific configuration where all tapes have the format just defined and the first tape contains the input. This is called the *start configuration* [2, p. 13].

definition *start-config* :: *nat* ⇒ *symbol list* ⇒ *config* **where**
start-config k xs ≡ (*0*, (⌊*xs*⌋, *0*) # *replicate* (*k* − *1*) (⌊[]⌋, *0*))

lemma *start-config-length*: *k* > *0* ⟹ ||*start-config k xs*|| = *k*
using *start-config-def contents-def* **by** *simp*

lemma *start-config1*:
assumes *cfg* = *start-config k xs* **and** *0* < *j* **and** *j* < *k* **and** *i* > *0*
shows (*cfg* <:> *j*) *i* = □
using *start-config-def contents-def assms* **by** *simp*

lemma *start-config2*:
assumes *cfg* = *start-config k xs* **and** *j* < *k*
shows (*cfg* <:> *j*) *0* = ▷
using *start-config-def contents-def assms* **by** (*cases 0* = *j*) *simp-all*

lemma *start-config3*:

assumes *cfg* = *start-config k xs* **and** *i* > *0* **and** *i* ≤ *length xs*
shows (*cfg* <:> *0*) *i* = *xs* ! (*i* − *1*)
using *start-config-def contents-def assms* **by** *simp*

lemma *start-config4*:
assumes *0* < *j* **and** *j* < *k*
shows *snd* (*start-config k xs*) ! *j* = (λ*i*. *if i* = *0 then* ▷ *else* □, *0*)
using *start-config-def contents-def assms* **by** *auto*

lemma *start-config-pos*: *j* < *k* ⟹ *start-config k zs* <#> *j* = *0*
using *start-config-def* **by** (*simp add*: *nth-Cons′*)

We call a symbol *proper* if it is neither the blank symbol nor the start symbol.

abbreviation *proper-symbols* :: *symbol list* ⇒ *bool* **where**
proper-symbols xs ≡ ∀ *i*<*length xs*. *xs* ! *i* > *Suc 0*

lemma *proper-symbols-append*:
assumes *proper-symbols xs* **and** *proper-symbols ys*
shows *proper-symbols* (*xs* @ *ys*)
using *assms prop-list-append* **by** (*simp add*: *nth-append*)

lemma *proper-symbols-ne0*: *proper-symbols xs* ⟹ ∀ *i*<*length xs*. *xs* ! *i* ≠ □
by *auto*

lemma *proper-symbols-ne1*: *proper-symbols xs* ⟹ ∀ *i*<*length xs*. *xs* ! *i* ≠ ▷
by *auto*

We call the symbols **0** and **1** *bit symbols*.

abbreviation *bit-symbols* :: *nat list* ⇒ *bool* **where**
bit-symbols xs ≡ ∀ *i*<*length xs*. *xs* ! *i* = **0** ∨ *xs* ! *i* = **1**

lemma *bit-symbols-append*:
assumes *bit-symbols xs* **and** *bit-symbols ys*
shows *bit-symbols* (*xs* @ *ys*)
using *assms prop-list-append* **by** (*simp add*: *nth-append*)

Basic facts about Turing machines

A Turing machine with alphabet G started on a symbol sequence over G will only ever have symbols from G on any of its tapes.

lemma *tape-alphabet*:
assumes *turing-machine k G M* **and** *symbols-lt G zs* **and** *j* < *k*
shows ((*execute M* (*start-config k zs*) *t*) <:> *j*) *i* < *G*
using *assms*(*3*)
proof (*induction t arbitrary*: *i j*)
case *0*
have *G* ≥ *2*
using *turing-machine-def assms*(*1*) **by** *simp*
then show *?case*
using *start-config-def contents-def 0 assms*(*2*) *start-config1 start-config2*
by (*smt* (*verit*) *One-nat-def Suc-1 Suc-lessD Suc-pred execute.simps*(*1*)
fst-conv lessI nat-less-le neq0-conv nth-Cons-0 snd-conv)
next
case (*Suc t*)
let *?cfg* = *execute M* (*start-config k zs*) *t*
have ∗: *execute M* (*start-config k zs*) (*Suc t*) = *exe M ?cfg*
by *simp*
show *?case*
proof (*cases fst ?cfg* ≥ *length M*)
case *True*
then have *execute M* (*start-config k zs*) (*Suc t*) = *?cfg*
using ∗ *exe-def* **by** *simp*

```
    then show ?thesis
      using Suc by simp
  next
    case False
    then have **: execute M (start-config k zs) (Suc t) = sem (M ! (fst ?cfg)) ?cfg
      using * exe-def by simp
    let ?rs = config-read ?cfg
    let ?cmd = M ! (fst ?cfg)
    let ?sas = ?cmd ?rs
    let ?cfg′ = sem ?cmd ?cfg
    have ∀ j<length ?rs. ?rs ! j < G
      using Suc assms(1) execute-num-tapes start-config-length read-abbrev read-length by auto
    moreover have len: length ?rs = k
      using assms(1) assms(3) execute-num-tapes start-config-def read-length by auto
    moreover have 2: turing-command k (length M) G ?cmd
      using assms(1) turing-machine-def False leI by simp
    ultimately have sas: ∀ j<length ?rs. ?sas [.] j < G
      using turing-command-def by simp
    have ?cfg′ <!> j = act (?sas [!] j) (?cfg <!> j)
      using Suc.prems 2 len read-length sem-snd turing-commandD(1) by metis
    then have ?cfg′ <:> j = (?cfg <:> j)(?cfg <#> j := ?sas [.] j)
      using act by simp
    then have (?cfg′ <:> j) i < G
      by (simp add: len Suc sas)
    then show ?thesis
      using ** by simp
  qed
qed

corollary read-alphabet:
  assumes turing-machine k G M and symbols-lt G zs
  shows ∀ i<k. config-read (execute M (start-config k zs) t) ! i < G
  using assms tape-alphabet execute-num-tapes start-config-length read-abbrev
  by simp

corollary read-alphabet′:
  assumes turing-machine k G M and symbols-lt G zs
  shows symbols-lt G (config-read (execute M (start-config k zs) t))
  using read-alphabet assms execute-num-tapes start-config-length read-length turing-machine-def
  by (metis neq0-conv not-numeral-le-zero)

corollary read-alphabet-set:
  assumes turing-machine k G M and symbols-lt G zs
  shows ∀ h∈set (config-read (execute M (start-config k zs) t)). h < G
  using read-alphabet′[OF assms] by (metis in-set-conv-nth)
```

The contents of the input tape never change.

```
lemma input-tape-constant:
  assumes turing-machine k G M and k = ||cfg||
  shows execute M cfg t <:> 0 = execute M cfg 0 <:> 0
proof (induction t)
  case 0
  then show ?case
    by simp
next
  case (Suc t)
  let ?cfg = execute M cfg t
  have 1: execute M cfg (Suc t) = exe M ?cfg
    by simp
  have 2: length (read (snd ?cfg)) = k
    using execute-num-tapes assms read-length by simp
  have k: k > 0
    using assms(1) turing-machine-def by simp
```

show *?case*
proof (*cases fst ?cfg* < *length M*)
case *True*
then have *3*: *turing-command k* (*length M*) *G* (*M* ! *fst ?cfg*)
using *turing-machine-def assms*(*1*) **by** *simp*
then have (*M* ! *fst ?cfg*) (*read* (*snd ?cfg*)) [.] *0* = *read* (*snd ?cfg*) ! *0*
using *turing-command-def 2 k* **by** *auto*
then have *4*: (*M* ! *fst ?cfg*) (*read* (*snd ?cfg*)) [.] *0* = *?cfg* <.> *0*
using *2 k read-abbrev read-length* **by** *auto*
have *execute M cfg* (*Suc t*) <:> *0* = *sem* (*M* ! *fst ?cfg*) *?cfg* <:> *0*
using *True exe-def* **by** *simp*
also have ... = *fst* (*act* (((*M* ! *fst ?cfg*) (*read* (*snd ?cfg*))) [!] *0*) (*?cfg* <!> *0*))
using *sem-snd 2 3 k read-length turing-commandD*(*1*) **by** *metis*
also have ... = (*?cfg* <:> *0*) ((*?cfg* <#> *0*):=(((*M* ! *fst ?cfg*) (*read* (*snd ?cfg*))) [.] *0*))
using *act* **by** *simp*
also have ... = (*?cfg* <:> *0*) ((*?cfg* <#> *0*):=*?cfg* <.> *0*)
using *4* **by** *simp*
also have ... = *?cfg* <:> *0*
by *simp*
finally have *execute M cfg* (*Suc t*) <:> *0* = *?cfg* <:> *0* **.**
then show *?thesis*
using *Suc* **by** *simp*
next
case *False*
then have *execute M cfg* (*Suc t*) = *?cfg*
using *exe-def* **by** *simp*
then show *?thesis*
using *Suc* **by** *simp*
qed
qed

A head position cannot be greater than the number of steps the machine has been running.

lemma *head-pos-le-time*:
assumes *turing-machine k G M* **and** *j* < *k*
shows *execute M* (*start-config k zs*) *t* <#> *j* ≤ *t*
proof (*induction t*)
case *0*
have *0* < *k*
using *assms*(*1*) *turing-machine-def* **by** *simp*
then have *execute M* (*start-config k zs*) *0* <#> *j* = *0*
using *start-config-def assms*(*2*) *start-config-pos* **by** *simp*
then show *?case*
by *simp*
next
case (*Suc t*)
have *: *execute M* (*start-config k zs*) (*Suc t*) = *exe M* (*execute M* (*start-config k zs*) *t*)
(**is** - = *exe M ?cfg*)
by *simp*
show *?case*
proof (*cases fst ?cfg* = *length M*)
case *True*
then have *execute M* (*start-config k zs*) (*Suc t*) = *?cfg*
using * *exe-def* **by** *simp*
then show *?thesis*
using *Suc* **by** *simp*
next
case *False*
then have *less*: *fst ?cfg* < *length M*
using *assms*(*1*) *turing-machine-def*
by (*simp add*: *start-config-def le-neq-implies-less turing-machine-execute-states*)
then have *exe M ?cfg* = *sem* (*M* ! (*fst ?cfg*)) *?cfg*
using *exe-def* **by** *simp*
moreover have *proper-command k* (*M* ! (*fst ?cfg*))

```
      using assms(1) turing-commandD(1) less turing-machine-def nth-mem by blast
    ultimately have exe M ?cfg <!> j = act (snd ((M ! (fst ?cfg)) (config-read ?cfg)) ! j) (?cfg <!> j)
      using assms(1,2) execute-num-tapes start-config-length sem-snd by auto
    then have exe M ?cfg <#> j ≤ Suc (?cfg <#> j)
      using act-pos-le-Suc assms(1,2) execute-num-tapes start-config-length by auto
    then show ?thesis
      using ∗ Suc.IH by simp
  qed
qed

lemma head-pos-le-halting-time:
  assumes turing-machine k G M
    and fst (execute M (start-config k zs) T) = length M
    and j < k
  shows execute M (start-config k zs) t <#> j ≤ T
  using assms execute-after-halting-ge[OF assms(2)] head-pos-le-time[OF assms(1,3)]
  by (metis nat-le-linear order-trans)
```

A tape cannot contain non-blank symbols at a position larger than the number of steps the Turing machine has been running, except on the input tape.

```
lemma blank-after-time:
  assumes i > t and j < k and 0 < j and turing-machine k G M
  shows (execute M (start-config k zs) t <:> j) i = □
  using assms(1)
proof (induction t)
  case 0
  have execute M (start-config k zs) 0 = start-config k zs
    by simp
  then show ?case
    using start-config1 assms turing-machine-def by simp
next
  case (Suc t)
  have k ≥ 2
    using assms(2,3) by simp
  let ?icfg = start-config k zs
  have ∗: execute M ?icfg (Suc t) = exe M (execute M ?icfg t)
    by simp
  show ?case
  proof (cases fst (execute M ?icfg t) ≥ length M)
    case True
    then have execute M ?icfg (Suc t) = execute M ?icfg t
      using ∗ exe-def by simp
    then show ?thesis
      using Suc by simp
  next
    case False
    then have execute M ?icfg (Suc t) <:> j = sem (M ! (fst (execute M ?icfg t))) (execute M ?icfg t) <:> j
        (is - = sem ?cmd ?cfg <:> j)
      using exe-lt-length ∗ by simp
    also have ... = fst (map (λ(a, tp). act a tp) (zip (snd (?cmd (read (snd ?cfg)))) (snd ?cfg)) ! j)
      using sem′ by simp
    also have ... = fst (act (snd (?cmd (read (snd ?cfg))) ! j) (snd ?cfg ! j))
      (is - = fst (act ?h (snd ?cfg ! j)))
    proof −
      have ||?cfg|| = k
        using assms(2) execute-num-tapes[OF assms(4)] start-config-length turing-machine-def
        by simp
      moreover have length (snd (?cmd (read (snd ?cfg)))) = k
        using assms(4) execute-num-tapes[OF assms(4)] start-config-length turing-machine-def
          read-length False turing-command-def wf-command-def
        by simp
      ultimately show ?thesis
        using assms by simp
```

qed
finally have *execute M ?icfg* (*Suc t*) *<:> j* = *fst* (*act ?h* (*snd ?cfg* ! *j*)) **.**
moreover have *i* ≠ *?cfg <#> j*
using *head-pos-le-time*[*OF assms*(*4*,*2*)] *Suc Suc-lessD leD* **by** *blast*
ultimately have (*execute M ?icfg* (*Suc t*) *<:> j*) *i* = *fst* (*?cfg <!> j*) *i*
using *act-changes-at-most-pos* **by** (*metis prod.collapse*)
then show *?thesis*
using *Suc Suc-lessD* **by** *presburger*
qed
qed

2.1.2 Computing a function

Turing machines are supposed to compute functions. The functions in question map bit strings to bit strings. We model such strings as lists of Booleans and denote the bits by $\mathbb{O}$ and $\mathbb{I}$.

type-synonym *string* = *bool list*

notation *False* (‹$\mathbb{O}$›) **and** *True* (‹$\mathbb{I}$›)

This keeps the more abstract level of computable functions separate from the level of concrete implementations as Turing machines, which can use an arbitrary alphabet. We use the term "string" only for bit strings, on which functions operate, and the terms "symbol sequence" or "symbols" for the things written on the tapes of Turing machines. We translate between the two levels in a straightforward way:

abbreviation *string-to-symbols* :: *string* ⇒ *symbol list* **where**
string-to-symbols x ≡ *map* (λ*b*. *if b then* **1** *else* **0**) *x*

abbreviation *symbols-to-string* :: *symbol list* ⇒ *string* **where**
symbols-to-string zs ≡ *map* (λ*z*. *z* = **1**) *zs*

proposition
string-to-symbols [$\mathbb{O}$, $\mathbb{I}$] = [**0**, **1**]
symbols-to-string [**0**, **1**] = [$\mathbb{O}$, $\mathbb{I}$]
by *simp-all*

lemma *bit-symbols-to-symbols*:
assumes *bit-symbols zs*
shows *string-to-symbols* (*symbols-to-string zs*) = *zs*
using *assms* **by** (*intro nth-equalityI*) *auto*

lemma *symbols-to-string-to-symbols*: *symbols-to-string* (*string-to-symbols x*) = *x*
by (*intro nth-equalityI*) *simp-all*

lemma *proper-symbols-to-symbols*: *proper-symbols* (*string-to-symbols zs*)
by *simp*

abbreviation *string-to-contents* :: *string* ⇒ (*nat* ⇒ *symbol*) **where**
string-to-contents x ≡
λ*i*. *if i* = *0 then* ▷ *else if i* ≤ *length x then* (*if x* ! (*i* − *1*) *then* **1** *else* **0**) *else* □

lemma *contents-string-to-contents*: *string-to-contents xs* = ⌊*string-to-symbols xs*⌋
using *contents-def* **by** *auto*

lemma *bit-symbols-to-contents*:
assumes *bit-symbols ns*
shows ⌊*ns*⌋ = *string-to-contents* (*symbols-to-string ns*)
using *assms bit-symbols-to-symbols contents-string-to-contents* **by** *simp*

Definition 1.3 in the textbook [2] says that for a Turing machine M to compute a function $f\colon \{\mathbb{O},\mathbb{I}\}^* \to \{\mathbb{O},\mathbb{I}\}^*$ on input x, "it halts with $f(x)$ written on its output tape." My initial interpretation of this phrase, and the one formalized below, was that the output is written *after* the start symbol ▷ in the same fashion as the input is given on the input tape. However after inspecting the Turing machine in Example 1.1, I now believe the more likely meaning is that the output *overwrites* the start symbol, although Example 1.1

precedes Definition 1.3 and might not be subject to it.

One advantage of the interpretation with start symbol intact is that the output tape can then be used unchanged as the input of another Turing machine, a property we exploit in Section 2.6. Otherwise one would have to find the start cell of the output tape and either copy the contents to another tape with start symbol or shift the string to the right and restore the start symbol. One way to find the start cell is to move the tape head left while "marking" the cells until one reaches an already marked cell, which can only happen when the head is in the start cell, where "moving left" does not actually move the head. This process will take time linear in the length of the output and thus will not change the asymptotic running time of the machine. Therefore the choice of interpretation is purely one of convenience.

definition *halts* :: *machine* ⇒ *config* ⇒ *bool* **where**
  halts M cfg ≡ ∃ *t*. *fst* (*execute M cfg t*) = *length M*

lemma *halts-impl-le-length*:
  assumes *halts M cfg*
  shows *fst* (*execute M cfg t*) ≤ *length M*
  using *assms execute-after-halting-ge′ halts-def* **by** (*metis linear*)

definition *running-time* :: *machine* ⇒ *config* ⇒ *nat* **where**
  running-time M cfg ≡ *LEAST t*. *fst* (*execute M cfg t*) = *length M*

lemma *running-timeD*:
  assumes *running-time M cfg* = *t* **and** *halts M cfg*
  shows *fst* (*execute M cfg t*) = *length M*
    and $\bigwedge t'$. $t' < t \Longrightarrow$ *fst* (*execute M cfg t′*) ≠ *length M*
  using *assms running-time-def halts-def*
    not-less-Least[*of - λt. fst* (*execute M cfg t*) = *length M*]
    LeastI[*of λt. fst* (*execute M cfg t*) = *length M*]
  by *auto*

definition *halting-config* :: *machine* ⇒ *config* ⇒ *config* **where**
  halting-config M cfg ≡ *execute M cfg* (*running-time M cfg*)

abbreviation *start-config-string* :: *nat* ⇒ *string* ⇒ *config* **where**
  start-config-string k x ≡ *start-config k* (*string-to-symbols x*)

Another, inconsequential, difference to the textbook definition is that we designate the second tape, rather than the last tape, as the output tape. This means that the indices for the input and output tape are fixed at 0 and 1, respectively, regardless of the total number of tapes. Next is our definition of a k-tape Turing machine M computing a function f in T-time:

definition *computes-in-time* :: *nat* ⇒ *machine* ⇒ (*string* ⇒ *string*) ⇒ (*nat* ⇒ *nat*) ⇒ *bool* **where**
  computes-in-time k M f T ≡ ∀ *x*.
    halts M (*start-config-string k x*) ∧
    running-time M (*start-config-string k x*) ≤ *T* (*length x*) ∧
    halting-config M (*start-config-string k x*) <:> *1* = *string-to-contents* (*f x*)

lemma *computes-in-time-mono*:
  assumes *computes-in-time k M f T* **and** $\bigwedge n$. $T\ n \le T'\ n$
  shows *computes-in-time k M f T′*
  using *assms computes-in-time-def halts-def running-time-def halting-config-def execute-after-halting-ge*
  by (*meson dual-order.trans*)

The definition of *computes-in-time* can be expressed with *transforms* as well, which will be more convenient for us.

lemma *halting-config-execute*:
  assumes *fst* (*execute M cfg t*) = *length M*
  shows *halting-config M cfg* = *execute M cfg t*
proof −
  have *1*: *t* ≥ *running-time M cfg*
    using *assms running-time-def* **by** (*simp add*: *Least-le*)
  then have *fst* (*halting-config M cfg*) = *length M*

using *assms LeastI*[*of* $\lambda t.$ *fst* (*execute M cfg t*) = *length M t*]
by (*simp add*: *halting-config-def running-time-def*)
then show *?thesis*
using *execute-after-halting-ge 1 halting-config-def* **by** *metis*
qed

lemma *transforms-halting-config*:
assumes *transforms M tps t tps'*
shows *halting-config M* (*0*, *tps*) = (*length M*, *tps'*)
using *assms transforms-def halting-config-def halting-config-execute transits-def*
by (*metis fst-eqD*)

lemma *computes-in-time-execute*:
assumes *computes-in-time k M f T*
shows *execute M* (*start-config-string k x*) (*T* (*length x*)) <:> *1* = *string-to-contents* (*f x*)
proof −
let *?t* = *running-time M* (*start-config-string k x*)
let *?cfg* = *start-config-string k x*
have *execute M ?cfg ?t* = *halting-config M ?cfg*
using *halting-config-def* **by** *simp*
then have *fst* (*execute M ?cfg ?t*) = *length M*
using *assms computes-in-time-def running-timeD*(*1*) **by** *blast*
moreover have *?t* ≤ *T* (*length x*)
using *computes-in-time-def assms* **by** *simp*
ultimately have *execute M ?cfg ?t* = *execute M ?cfg* (*T* (*length x*))
using *execute-after-halting-ge* **by** *presburger*
moreover have *execute M ?cfg ?t* <:> *1* = *string-to-contents* (*f x*)
using *computes-in-time-def halting-config-execute assms halting-config-def* **by** *simp*
ultimately show *?thesis*
by *simp*
qed

lemma *transforms-running-time*:
assumes *transforms M tps t tps'*
shows *running-time M* (*0*, *tps*) ≤ *t*
using *running-time-def transforms-def transits-def*
by (*smt* (*verit*) *Least-le*[*of* - *t*] *assms execute-after-halting-ge fst-conv*)

This is the alternative characterization of *computes-in-time*:

lemma *computes-in-time-alt*:
computes-in-time k M f T =
(∀ *x*. ∃ *tps*.
tps ::: *1* = *string-to-contents* (*f x*) ∧
transforms M (*snd* (*start-config-string k x*)) (*T* (*length x*)) *tps*)
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
proof
fix *x* :: *string*
let *?cfg* = *start-config-string k x*
assume *computes-in-time k M f T*
then have
1: *halts M ?cfg* **and**
2: *running-time M ?cfg* ≤ *T* (*length x*) **and**
3: *halting-config M ?cfg* <:> *1* = *string-to-contents* (*f x*)
using *computes-in-time-def* **by** *simp-all*
define *cfg* **where** *cfg* = *halting-config M ?cfg*
then have *transits M ?cfg* (*T* (*length x*)) *cfg*
using *2 halting-config-def transits-def* **by** *auto*
then have *transforms M* (*snd ?cfg*) (*T* (*length x*)) (*snd cfg*)
using *transits-def transforms-def start-config-def*
by (*metis* (*no-types*, *lifting*) *1 cfg-def halting-config-def prod.collapse running-timeD*(*1*) *snd-conv*)
moreover have *snd cfg* ::: *1* = *string-to-contents* (*f x*)

    using *cfg-def 3* **by** *simp*
  ultimately show $\exists$ *tps. tps* ::: *1* = *string-to-contents* (*f x*) $\wedge$
      transforms M (*snd* (*start-config-string k x*)) (*T* (*length x*)) *tps*
    by *auto*
 qed
 show *?rhs* $\Longrightarrow$ *?lhs*
  unfolding *computes-in-time-def*
 proof
  assume *rhs*: *?rhs*
  fix *x* :: *string*
  let *?cfg* = *start-config-string k x*
  obtain *tps* **where** *tps*:
    tps ::: *1* = *string-to-contents* (*f x*)
    transforms M (*snd ?cfg*) (*T* (*length x*)) *tps*
    using *rhs* **by** *auto*
  then have *transits M ?cfg* (*T* (*length x*)) (*length M*, *tps*)
    using *transforms-def start-config-def* **by** *simp*
  then have *1*: *halts M ?cfg*
    using *halts-def transits-def* **by** (*metis fst-eqD*)
  moreover have *2*: *running-time M ?cfg* $\leq$ *T* (*length x*)
    using *tps*(*2*) *transforms-running-time start-config-def* **by** *simp*
  moreover have *3*: *halting-config M ?cfg* <:> *1* = *string-to-contents* (*f x*)
  proof −
    have *halting-config M ?cfg* = (*length M*, *tps*)
      using *transforms-halting-config*[*OF tps*(*2*)] *start-config-def* **by** *simp*
    then show *?thesis*
      using *tps*(*1*) **by** *simp*
  qed
   ultimately show *halts M ?cfg* $\wedge$ *running-time M ?cfg* $\leq$ *T* (*length x*) $\wedge$ *halting-config M ?cfg* <:> *1* = *string-to-contents* (*f x*)
    by *simp*
 qed
qed

lemma *computes-in-timeD*:
 fixes *x*
 assumes *computes-in-time k M f T*
 shows $\exists$ *tps. tps* ::: *1* = *string-to-contents* (*f x*) $\wedge$
     transforms M (*snd* (*start-config k* (*string-to-symbols x*))) (*T* (*length x*)) *tps*
 using *assms computes-in-time-alt* **by** *simp*

lemma *computes-in-timeI* [*intro*]:
 assumes $\bigwedge$*x*. $\exists$ *tps. tps* ::: *1* = *string-to-contents* (*f x*) $\wedge$
    transforms M (*snd* (*start-config k* (*string-to-symbols x*))) (*T* (*length x*)) *tps*
 shows *computes-in-time k M f T*
 using *assms computes-in-time-alt* **by** *simp*

As an example, the function mapping every string to the empty string is computable within any time bound by the empty Turing machine.

lemma *computes-Nil-empty*:
 assumes *k* $\geq$ *2*
 shows *computes-in-time k* [] (λ*x*. []) *T*
proof
 fix *x* :: *string*
 let *?tps* = *snd* (*start-config-string k x*)
 let *?f* = λ*x*. []
 have *?tps* ::: *1* = *string-to-contents* (*?f x*)
  using *start-config4 assms* **by** *auto*
 moreover have *transforms* [] *?tps* (*T* (*length x*)) *?tps*
  using *transforms-Nil transforms-monotone* **by** *blast*
 ultimately show $\exists$ *tps. tps* ::: *1* = *string-to-contents* (*?f x*) $\wedge$ *transforms* [] *?tps* (*T* (*length x*)) *tps*
  by *auto*
qed

2.1.3 Pairing strings

In order to define the computability of functions with two arguments, we need a way to encode a pair of strings as one string. The idea is to write the two strings with a separator, for example, $\mathbb{OIOO}\#\mathbb{IIIO}$ and then encode every symbol $\mathbb{O}, \mathbb{I}, \#$ by two bits from $\{\mathbb{O}, \mathbb{I}\}$. We slightly deviate from Arora and Barak's encoding [2, p. 2] and map $\mathbb{O}$ to $\mathbb{OO}$, $\mathbb{I}$ to $\mathbb{OI}$, and $\#$ to $\mathbb{II}$, the idea being that the first bit signals whether the second bit is to be taken literally or as a special character. Our example turns into $\mathbb{OOOIOOOOIIIOIOIOIOO}$.

abbreviation *bitenc* :: *string* ⇒ *string* **where**
  bitenc x ≡ *concat* (*map* (λ*h*. [𝕆, *h*]) *x*)

definition *string-pair* :: *string* ⇒ *string* ⇒ *string* (‹⟨-, -⟩›) **where**
  ⟨*x*, *y*⟩ ≡ *bitenc x* @ [𝕀, 𝕀] @ *bitenc y*

Our example:

proposition ⟨[𝕆, 𝕀, 𝕆, 𝕆], [𝕀, 𝕀, 𝕀, 𝕆]⟩ = [𝕆, 𝕆, 𝕆, 𝕀, 𝕆, 𝕆, 𝕆, 𝕆, 𝕀, 𝕀, 𝕆, 𝕀, 𝕆, 𝕀, 𝕆, 𝕀, 𝕆, 𝕆]
  using *string-pair-def* **by** *simp*

lemma *length-string-pair*: *length* ⟨*x*, *y*⟩ = *2* ∗ *length x* + *2* ∗ *length y* + *2*
proof −
  have *length* (*concat* (*map* (λ*h*. [𝕆, *h*]) *z*)) = *2* ∗ *length z* **for** *z*
    by (*induction z*) *simp-all*
  then show *?thesis*
    using *string-pair-def* **by** *simp*
qed

lemma *length-bitenc*: *length* (*bitenc z*) = *2* ∗ *length z*
  by (*induction z*) *simp-all*

lemma *bitenc-nth*:
  assumes *i* < *length zs*
  shows *bitenc zs* ! (*2* ∗ *i*) = 𝕆
    and *bitenc zs* ! (*2* ∗ *i* + *1*) = *zs* ! *i*
proof −
  let *?f* = λ*h*. [𝕆, *h*]
  let *?xs* = *concat* (*map ?f zs*)

  have *eqtake*: *bitenc* (*take i zs*) = *take* (*2* ∗ *i*) (*bitenc zs*)
      if *i* ≤ *length zs* **for** *i zs*
  proof −
    have *take* (*2* ∗ *i*) (*bitenc zs*) = *take* (*2* ∗ *i*) (*bitenc* (*take i zs* @ *drop i zs*))
      by *simp*
    then have *take* (*2* ∗ *i*) (*bitenc zs*) = *take* (*2* ∗ *i*) (*bitenc* (*take i zs*) @ (*bitenc* (*drop i zs*)))
      by (*metis concat-append map-append*)
    then show *?thesis*
      using *length-bitenc that* **by** *simp*
  qed

  have *eqdrop*: *bitenc* (*drop i zs*) = *drop* (*2* ∗ *i*) (*bitenc zs*)
      if *i* < *length zs* **for** *i*
  proof −
    have *drop* (*2* ∗ *i*) (*bitenc zs*) = *drop* (*2* ∗ *i*) (*bitenc* (*take i zs* @ *drop i zs*))
      by *simp*
    then have *drop* (*2* ∗ *i*) (*bitenc zs*) = *drop* (*2* ∗ *i*) (*bitenc* (*take i zs*) @ *bitenc* (*drop i zs*))
      by (*metis concat-append map-append*)
    then show *?thesis*
      using *length-bitenc that* **by** *simp*
  qed

  have *take2*: *take 2* (*drop* (*2* ∗ *i*) (*bitenc zs*)) = *?f* (*zs* ! *i*) **if** *i* < *length zs* **for** *i*
  proof −
    have *1*: *1* ≤ *length* (*drop i zs*)

```
      using that by simp
    have take 2 (drop (2∗i) (bitenc zs)) = take 2 (bitenc (drop i zs))
      using that eqdrop by simp
    also have ... = bitenc (take 1 (drop i zs))
      using 1 eqtake by simp
    also have ... = bitenc [zs ! i]
      using that by (metis Cons-nth-drop-Suc One-nat-def take0 take-Suc-Cons)
    also have ... = ?f (zs ! i)
      by simp
    finally show ?thesis .
  qed

  show bitenc zs ! (2 ∗ i) = 𝟘
  proof −
    have bitenc zs ! (2 ∗ i) = drop (2 ∗ i) (bitenc zs) ! 0
      using assms drop0 length-bitenc by simp
    also have ... = take 2 (drop (2 ∗ i) (bitenc zs)) ! 0
      using eqdrop by simp
    also have ... = ?f (zs ! i) ! 0
      using assms take2 by simp
    also have ... = 𝟘
      by simp
    finally show ?thesis .
  qed

  show bitenc zs ! (2∗i + 1) = zs ! i
  proof −
    have bitenc zs ! (2∗i+1) = drop (2 ∗ i) (bitenc zs) ! 1
      using assms length-bitenc by simp
    also have ... = take 2 (drop (2∗i) (bitenc zs)) ! 1
      using eqdrop by simp
    also have ... = ?f (zs ! i) ! 1
      using assms(1) take2 by simp
    also have ... = zs ! i
      by simp
    finally show ?thesis .
  qed
qed

lemma string-pair-first-nth:
  assumes i < length x
  shows ⟨x, y⟩ ! (2 ∗ i) = 𝟘
    and ⟨x, y⟩ ! (2 ∗ i + 1) = x ! i
proof −
  have ⟨x, y⟩ ! (2∗i) = concat (map (λh. [𝟘, h]) x) ! (2∗i)
    using string-pair-def length-bitenc by (simp add: assms nth-append)
  then show ⟨x, y⟩ ! (2 ∗ i) = 𝟘
    using bitenc-nth(1) assms by simp
  have 2 ∗ i + 1 < 2 ∗ length x
    using assms by simp
  then have ⟨x, y⟩ ! (2∗i+1) = concat (map (λh. [𝟘, h]) x) ! (2∗i+1)
    using string-pair-def length-bitenc[of x] assms nth-append
    by force
  then show ⟨x, y⟩ ! (2 ∗ i + 1) = x ! i
    using bitenc-nth(2) assms by simp
qed

lemma string-pair-sep-nth:
  shows ⟨x, y⟩ ! (2 ∗ length x) = 𝟙
    and ⟨x, y⟩ ! (2 ∗ length x + 1) = 𝟙
  using string-pair-def length-bitenc
  by (metis append-Cons nth-append-length) (simp add: length-bitenc nth-append string-pair-def)
```

```
lemma string-pair-second-nth:
  assumes i < length y
  shows ⟨x, y⟩ ! (2 ∗ length x + 2 + 2 ∗ i) = 𝕆
    and ⟨x, y⟩ ! (2 ∗ length x + 2 + 2 ∗ i + 1) = y ! i
proof −
  have ⟨x, y⟩ ! (2 ∗ length x + 2 + 2∗i) = concat (map (λh. [𝕆, h]) y) ! (2∗i)
    using string-pair-def length-bitenc by (simp add: assms nth-append)
  then show ⟨x, y⟩ ! (2 ∗ length x + 2 + 2 ∗ i) = 𝕆
    using bitenc-nth(1) assms by simp
  have 2 ∗ i + 1 < 2 ∗ length y
    using assms by simp
  then have ⟨x, y⟩ ! (2 ∗ length x + 2 + 2∗i+1) = concat (map (λh. [𝕆, h]) y) ! (2∗i+1)
    using string-pair-def length-bitenc[of x] assms nth-append
    by force
  then show ⟨x, y⟩ ! (2 ∗ length x + 2 + 2 ∗ i + 1) = y ! i
    using bitenc-nth(2) assms by simp
qed

lemma string-pair-inj:
  assumes ⟨x1, y1⟩ = ⟨x2, y2⟩
  shows x1 = x2 ∧ y1 = y2
proof
  show x1 = x2
  proof (rule ccontr)
    assume neq: x1 ≠ x2
    consider length x1 = length x2 | length x1 < length x2 | length x1 > length x2
      by linarith
    then show False
    proof (cases)
      case 1
      then obtain i where i: i < length x1 x1 ! i ≠ x2 ! i
        using neq list-eq-iff-nth-eq by blast
      then have ⟨x1, y1⟩ ! (2 ∗ i + 1) = x1 ! i and ⟨x2, y2⟩ ! (2 ∗ i + 1) = x2 ! i
        using 1 string-pair-first-nth by simp-all
      then show False
        using assms i(2) by simp
    next
      case 2
      let ?i = length x1
      have ⟨x1, y1⟩ ! (2 ∗ ?i) = 𝕀
        using string-pair-sep-nth by simp
      moreover have ⟨x2, y2⟩ ! (2 ∗ ?i) = 𝕆
        using string-pair-first-nth 2 by simp
      ultimately show False
        using assms by simp
    next
      case 3
      let ?i = length x2
      have ⟨x2, y2⟩ ! (2 ∗ ?i) = 𝕀
        using string-pair-sep-nth by simp
      moreover have ⟨x1, y1⟩ ! (2 ∗ ?i) = 𝕆
        using string-pair-first-nth 3 by simp
      ultimately show False
        using assms by simp
    qed
  qed
  then have len-x-eq: length x1 = length x2
    by simp
  then have len-y-eq: length y1 = length y2
    using assms length-string-pair
    by (smt (verit) Suc-1 Suc-mult-cancel1 add-left-imp-eq add-right-cancel)
  show y1 = y2
  proof (rule ccontr)
```

assume *neq*: $y1 \neq y2$
then obtain *i* **where** *i*: $i <$ *length y1 y1* ! $i \neq$ *y2* ! *i*
using *list-eq-iff-nth-eq len-y-eq* **by** *blast*
then have $\langle x1, y1\rangle$! $(2 * \mathit{length}\ x1 + 2 + 2 * i + 1) = y1$! *i* **and**
$\langle x2, y2\rangle$! $(2 * \mathit{length}\ x2 + 2 + 2 * i + 1) = y2$! *i*
using *string-pair-second-nth len-y-eq* **by** *simp-all*
then show *False*
using *assms i*(*2*) *len-x-eq* **by** *simp*
qed
qed

Turing machines have to deal with pairs of symbol sequences rather than strings.

abbreviation *pair* :: *string* ⇒ *string* ⇒ *symbol list* (‹⟨-; -⟩›) **where**
⟨*x*; *y*⟩ ≡ *string-to-symbols* ⟨*x*, *y*⟩

lemma *symbols-lt-pair*: *symbols-lt 4* ⟨*x*; *y*⟩
by *simp*

lemma *length-pair*: *length* ⟨*x*; *y*⟩ = *2* ∗ *length x* + *2* ∗ *length y* + *2*
by (*simp add*: *length-string-pair*)

lemma *pair-inj*:
assumes ⟨*x1*; *y1*⟩ = ⟨*x2*; *y2*⟩
shows *x1* = *x2* ∧ *y1* = *y2*
using *string-pair-inj assms symbols-to-string-to-symbols* **by** *metis*

2.1.4 Big-Oh and polynomials

The Big-Oh notation is standard [2, Definition 0.2]. It can be defined with *c* ranging over real or natural numbers. We choose natural numbers for simplicity.

definition *big-oh* :: (*nat* ⇒ *nat*) ⇒ (*nat* ⇒ *nat*) ⇒ *bool* **where**
big-oh g f ≡ ∃ *c m*. ∀ *n*>*m*. *g n* ≤ *c* ∗ *f n*

Some examples:

proposition *big-oh* (λ*n*. *n*) (λ*n*. *n*)
using *big-oh-def* **by** *auto*

proposition *big-oh* (λ*n*. *n*) (λ*n*. *n* ∗ *n*)
using *big-oh-def* **by** *auto*

proposition *big-oh* (λ*n*. *42* ∗ *n*) (λ*n*. *n* ∗ *n*)
proof−
have ∀ *n*>*0*::*nat*. *42* ∗ *n* ≤ *42* ∗ *n* ∗ *n*
by *simp*
then have ∃ (*c*::*nat*)>*0*. ∀ *n*>*0*. *42* ∗ *n* ≤ *c* ∗ *n* ∗ *n*
using *zero-less-numeral* **by** *blast*
then show *?thesis*
using *big-oh-def* **by** *auto*
qed

proposition ¬ *big-oh* (λ*n*. *n* ∗ *n*) (λ*n*. *n*) (**is** ¬ *big-oh ?g ?f*)
proof
assume *big-oh* (λ*n*. *n* ∗ *n*) (λ*n*. *n*)
then obtain *c m* **where** ∀ *n*>*m*. *?g n* ≤ *c* ∗ *?f n*
using *big-oh-def* **by** *auto*
then have *1*: ∀ *n*>*m*. *n* ∗ *n* ≤ *c* ∗ *n*
by *auto*
define *nn* **where** *nn* = *max* (*m* + *1*) (*c* + *1*)
then have *2*: *nn* > *m*
by *simp*
then have *nn* ∗ *nn* > *c* ∗ *nn*
by (*simp add*: *nn-def max-def*)

with *1 2* **show** *False*
using *not-le* **by** *blast*
qed

Some lemmas helping with polynomial upper bounds.

lemma *pow-mono*:
fixes *n d1 d2* :: *nat*
assumes $d1 \leq d2$ **and** $n > 0$
shows n ^ $d1 \leq n$ ^ $d2$
using *assms* **by** (*simp add*: *Suc-leI power-increasing*)

lemma *pow-mono'*:
fixes *n d1 d2* :: *nat*
assumes $d1 \leq d2$ **and** $0 < d1$
shows n ^ $d1 \leq n$ ^ $d2$
using *assms* **by** (*metis dual-order.eq-iff less-le-trans neq0-conv pow-mono power-eq-0-iff*)

lemma *linear-le-pow*:
fixes *n d1* :: *nat*
assumes $0 < d1$
shows $n \leq n$ ^ $d1$
using *assms* **by** (*metis One-nat-def gr-implies-not0 le-less-linear less-Suc0 self-le-power*)

The next definition formalizes the phrase "polynomially bounded" and the term "polynomial" in "polynomial running-time". This is often written "$f(n) = n^{O(1)}$" (for example, Arora and Barak [2, Example 0.3]).

definition *big-oh-poly* :: (*nat* ⇒ *nat*) ⇒ *bool* **where**
big-oh-poly f ≡ ∃ *d*. *big-oh f* (λ*n*. *n* ^ *d*)

lemma *big-oh-poly*: *big-oh-poly f* ⟷ (∃ *d c* n_0. ∀ $n > n_0$. *f n* ≤ *c* ∗ *n* ^ *d*)
using *big-oh-def big-oh-poly-def* **by** *auto*

lemma *big-oh-polyI*:
assumes ⋀*n*. $n > n_0$ ⟹ *f n* ≤ *c* ∗ *n* ^ *d*
shows *big-oh-poly f*
using *assms big-oh-poly* **by** *auto*

lemma *big-oh-poly-const*: *big-oh-poly* (λ*n*. *c*)
proof −
let *?c* = *max 1 c*
have (λ*n*. *c*) *n* ≤ *?c* ∗ *n* ^ *1* **if** $n > 0$ **for** *n*
proof −
have *c* ≤ *n* ∗ *?c*
by (*metis* (*no-types*) *le-square max.cobounded2 mult.assoc mult-le-mono nat-mult-le-cancel-disj that*)
then show *?thesis*
by (*simp add*: *mult.commute*)
qed
then show *?thesis*
using *big-oh-polyI*[*of 0 - ?c*] **by** *simp*
qed

lemma *big-oh-poly-poly*: *big-oh-poly* (λ*n*. *n* ^ *d*)
using *big-oh-polyI*[*of 0 - 1 d*] **by** *simp*

lemma *big-oh-poly-id*: *big-oh-poly* (λ*n*. *n*)
using *big-oh-poly-poly*[*of 1*] **by** *simp*

lemma *big-oh-poly-le*:
assumes *big-oh-poly f* **and** ⋀*n*. *g n* ≤ *f n*
shows *big-oh-poly g*
using *assms big-oh-polyI* **by** (*metis big-oh-poly le-trans*)

lemma *big-oh-poly-sum*:

```
  assumes big-oh-poly f1 and big-oh-poly f2
  shows big-oh-poly (λn. f1 n + f2 n)
proof−
  obtain d1 c1 m1 where 1: ∀n>m1. f1 n ≤ c1 ∗ n ^ d1
    using big-oh-poly assms(1) by blast
  obtain d2 c2 m2 where 2: ∀n>m2. f2 n ≤ c2 ∗ n ^ d2
    using big-oh-poly assms(2) by blast
  let ?f3 = λn. f1 n + f2 n
  let ?c3 = max c1 c2
  let ?m3 = max m1 m2
  let ?d3 = max d1 d2
  have ∀n>?m3. f1 n ≤ ?c3 ∗ n ^ d1
    using 1 by (simp add: max.coboundedI1 nat-mult-max-left)
  moreover have ∀n>?m3. n ^ d1 ≤ n^?d3
    using pow-mono by simp
  ultimately have ∗: ∀n>?m3. f1 n ≤ ?c3 ∗ n^?d3
    using order-subst1 by fastforce
  have ∀n>?m3. f2 n ≤ ?c3 ∗ n ^ d2
    using 2 by (simp add: max.coboundedI2 nat-mult-max-left)
  moreover have ∀n>?m3. n ^ d2 ≤ n ^ ?d3
    using pow-mono by simp
  ultimately have ∀n>?m3. f2 n ≤ ?c3 ∗ n ^ ?d3
    using order-subst1 by fastforce
  then have ∀n>?m3. f1 n + f2 n ≤ ?c3 ∗ n ^ ?d3 + ?c3 ∗ n ^ ?d3
    using ∗ by fastforce
  then have ∀n>?m3. f1 n + f2 n ≤ 2 ∗ ?c3 ∗ n ^ ?d3
    by auto
  then have ∃d c m. ∀n>m. ?f3 n ≤ c ∗ n ^ d
    by blast
  then show ?thesis
    using big-oh-poly by simp
qed

lemma big-oh-poly-prod:
  assumes big-oh-poly f1 and big-oh-poly f2
  shows big-oh-poly (λn. f1 n ∗ f2 n)
proof−
  obtain d1 c1 m1 where 1: ∀n>m1. f1 n ≤ c1 ∗ n ^ d1
    using big-oh-poly assms(1) by blast
  obtain d2 c2 m2 where 2: ∀n>m2. f2 n ≤ c2 ∗ n ^ d2
    using big-oh-poly assms(2) by blast
  let ?f3 = λn. f1 n ∗ f2 n
  let ?c3 = max c1 c2
  let ?m3 = max m1 m2
  have ∀n>?m3. f1 n ≤ ?c3 ∗ n ^ d1
    using 1 by (simp add: max.coboundedI1 nat-mult-max-left)
  moreover have ∀n>?m3. n ^ d1 ≤ n ^ d1
    using pow-mono by simp
  ultimately have ∗: ∀n>?m3. f1 n ≤ ?c3 ∗ n ^ d1
    using order-subst1 by fastforce
  have ∀n>?m3. f2 n ≤ ?c3 ∗ n ^ d2
    using 2 by (simp add: max.coboundedI2 nat-mult-max-left)
  moreover have ∀n>?m3. n ^ d2 ≤ n ^ d2
    using pow-mono by simp
  ultimately have ∀n>?m3. f2 n ≤ ?c3 ∗ n ^ d2
    using order-subst1 by fastforce
  then have ∀n>?m3. f1 n ∗ f2 n ≤ ?c3 ∗ n ^ d1 ∗ ?c3 ∗ n ^ d2
    using ∗ mult-le-mono by (metis mult.assoc)
  then have ∀n>?m3. f1 n ∗ f2 n ≤ ?c3 ∗ ?c3 ∗ n ^ d1 ∗ n ^ d2
    by (simp add: semiring-normalization-rules(16))
  then have ∀n>?m3. f1 n ∗ f2 n ≤ ?c3 ∗ ?c3 ∗ n ^(d1 + d2)
    by (simp add: mult.assoc power-add)
  then have ∃d c m. ∀n>m. ?f3 n ≤ c ∗ n ^ d
```

```
    by blast
  then show ?thesis
    using big-oh-poly by simp
qed

lemma big-oh-poly-offset:
  assumes big-oh-poly f
  shows ∃ b c d. d > 0 ∧ (∀ n. f n ≤ b + c ∗ n ^ d)
proof –
  obtain d c m where dcm: ∀ n>m. f n ≤ c ∗ n ^ d
    using assms big-oh-poly by auto
  have ∗: f n ≤ c ∗ n ^ Suc d if n > m for n
  proof –
    have n > 0
      using that by simp
    then have n ^ d ≤ n ^ Suc d
      by simp
    then have c ∗ n ^ d ≤ c ∗ n ^ Suc d
      by simp
    then show f n ≤ c ∗ n ^ Suc d
      using dcm order-trans that by blast
  qed
  define b :: nat where b = Max {f n | n. n ≤ m}
  then have y ≤ b if y ∈ {f n | n. n ≤ m} for y
    using that by simp
  then have f n ≤ b if n ≤ m for n
    using that by auto
  then have f n ≤ b + c ∗ n ^ Suc d for n
    using ∗ by (meson trans-le-add1 trans-le-add2 verit-comp-simplify1(3))
  then show ?thesis
    using ∗ dcm(1) by blast
qed

lemma big-oh-poly-composition:
  assumes big-oh-poly f1 and big-oh-poly f2
  shows big-oh-poly (f2 ∘ f1)
proof–
  obtain d1 c1 m1 where 1: ∀ n>m1. f1 n ≤ c1 ∗ n ^ d1
    using big-oh-poly assms(1) by blast
  obtain d2 c2 b where 2: ∀ n. f2 n ≤ b + c2 ∗ n ^ d2
    using big-oh-poly-offset assms(2) by blast
  define c where c = c2 ∗ c1 ^ d2
  have 3: ∀ n>m1. f1 n ≤ c1 ∗ n ^ d1
    using 1 by simp
  have ∀ n>m1. f2 n ≤ b + c2 ∗ n ^ d2
    using 2 by simp
  { fix n
    assume n > m1
    then have 4: (f1 n) ^ d2 ≤ (c1 ∗ n ^ d1) ^ d2
      using 3 by (simp add: power-mono)
    have f2 (f1 n) ≤ b + c2 ∗ (f1 n) ^ d2
      using 2 by simp
    also have ... ≤ b + c2 ∗ (c1 ∗ n ^ d1) ^ d2
      using 4 by simp
    also have ... = b + c2 ∗ c1 ^ d2 ∗ n ^(d1 ∗ d2)
      by (simp add: power-mult power-mult-distrib)
    also have ... = b + c ∗ n ^(d1 ∗ d2)
      using c-def by simp
    also have ... ≤ b ∗ n ^(d1 ∗ d2) + c ∗ n ^(d1 ∗ d2)
      using ‹n > m1› by simp
    also have ... ≤ (b + c) ∗ n ^(d1 ∗ d2)
      by (simp add: comm-semiring-class.distrib)
    finally have f2 (f1 n) ≤ (b + c) ∗ n ^(d1 ∗ d2) .
```

```
  }
  then show ?thesis
    using big-oh-polyI[of m1 - b + c d1 * d2] by simp
qed

lemma big-oh-poly-pow:
  fixes f :: nat ⇒ nat and d :: nat
  assumes big-oh-poly f
  shows big-oh-poly (λn. f n ^ d)
proof –
  let ?g = λn. n ^ d
  have big-oh-poly ?g
    using big-oh-poly-poly by simp
  moreover have (λn. f n ^ d) = ?g ∘ f
    by auto
  ultimately show ?thesis
    using assms big-oh-poly-composition by simp
qed
```

The textbook does not give an explicit definition of polynomials. It treats them as functions between natural numbers. So assuming the coefficients are natural numbers too, seems natural. We justify this choice when defining $\mathcal{NP}$ in Section 3.1.

```
definition polynomial :: (nat ⇒ nat) ⇒ bool where
  polynomial f ≡ ∃ cs. ∀ n. f n = (∑ i←[0..<length cs]. cs ! i * n ^ i)

lemma const-polynomial: polynomial (λ-. c)
proof –
  let ?cs = [c]
  have ∀ n. (λ-. c) n = (∑ i←[0..<length ?cs]. ?cs ! i * n ^ i)
    by simp
  then show ?thesis
    using polynomial-def by blast
qed

lemma polynomial-id: polynomial id
proof –
  let ?cs = [0, 1::nat]
  have ∀ n::nat. id n = (∑ i←[0..<length ?cs]. ?cs ! i * n ^ i)
    by simp
  then show ?thesis
    using polynomial-def by blast
qed

lemma big-oh-poly-polynomial:
  fixes f :: nat ⇒ nat
  assumes polynomial f
  shows big-oh-poly f
proof –
  have big-oh-poly (λn. (∑ i←[0..<length cs]. cs ! i * n ^ i)) for cs
  proof (induction length cs arbitrary: cs)
    case 0
    then show ?case
      using big-oh-poly-const by simp
  next
    case (Suc len)
    let ?cs = butlast cs
    have len: length ?cs = len
      using Suc by simp
    {
      fix n :: nat
      have (∑ i←[0..<length cs]. cs ! i * n ^ i) = (∑ i←[0..<Suc len]. cs ! i * n ^ i)
        using Suc by simp
```

```
      also have ... = (∑ i←[0..<len]. cs ! i * n ^ i) + cs ! len * n ^ len
        using Suc(2)
        by (metis (mono-tags, lifting) Nat.add-0-right list.simps(8) list.simps(9) map-append
          sum-list.Cons sum-list.Nil sum-list-append upt-Suc zero-le)
      also have ... = (∑ i←[0..<len]. ?cs ! i * n ^ i) + cs ! len * n ^ len
        using Suc(2) len by (metis (no-types, lifting) atLeastLessThan-iff map-eq-conv nth-butlast set-upt)
      finally have (∑ i←[0..<length cs]. cs ! i * n ^ i) = (∑ i←[0..<len]. ?cs ! i * n ^ i) + cs ! len * n ^ len .
    }
    then have (λn. ∑ i←[0..<length cs]. cs ! i * n ^ i) = (λn. (∑ i←[0..<len]. ?cs ! i * n ^ i) + cs ! len * n ^
len)
      by simp
    moreover have big-oh-poly (λn. cs ! len * n ^ len)
      using big-oh-poly-poly big-oh-poly-prod big-oh-poly-const by simp
    moreover have big-oh-poly (λn. (∑ i←[0..<len]. ?cs ! i * n ^ i))
      using Suc len by blast
    ultimately show big-oh-poly (λn. ∑ i←[0..<length cs]. cs ! i * n ^ i)
      using big-oh-poly-sum by simp
  qed
  moreover obtain cs where f = (λn. (∑ i←[0..<length cs]. cs ! i * n ^ i))
    using assms polynomial-def by blast
  ultimately show ?thesis
    by simp
qed
```

2.2 Increasing the alphabet or the number of tapes

For technical reasons it is sometimes necessary to add tapes to a machine or to formally enlarge its alphabet such that it matches another machine's tape number or alphabet size without changing the behavior of the machine. The primary use of this is when composing machines with unequal alphabets or tape numbers (see Section 2.6).

2.2.1 Enlarging the alphabet

A Turing machine over alphabet G is not necessarily a Turing machine over a larger alphabet $G' > G$ because reading a symbol in $\{G, \dots, G' - 1\}$ the TM may write a symbol $\geq G'$. This is easy to remedy by modifying the TM to do nothing when it reads a symbol $\geq G$. It then formally satisfies the alphabet restriction property of Turing commands. This is rather crude, because the new TM loops infinitely on encountering a "forbidden" symbol, but it is good enough for our purposes.

The next function performs this transformation on a TM M over alphabet G. The resulting machine is a Turing machine for every alphabet size $G' \geq G$.

```
definition enlarged :: nat ⇒ machine ⇒ machine where
  enlarged G M ≡ map (λcmd rs. if symbols-lt G rs then cmd rs else (0, map (λr. (r, Stay)) rs)) M

lemma length-enlarged: length (enlarged G M) = length M
  using enlarged-def by simp

lemma enlarged-nth:
  assumes symbols-lt G gs and i < length M
  shows (M ! i) gs = (enlarged G M ! i) gs
  using assms enlarged-def by simp

lemma enlarged-write:
  assumes length gs = k and i < length M and turing-machine k G M
  shows length (snd ((M ! i) gs)) = length (snd ((enlarged G M ! i) gs))
proof (cases symbols-lt G gs)
  case True
  then show ?thesis
    using assms enlarged-def by simp
next
  case False
```

```
  then have (enlarged G M ! i) gs = (0, map (λr. (r, Stay)) gs)
    using assms enlarged-def by auto
  then show ?thesis
    using assms turing-commandD(1) turing-machine-def by (metis length-map nth-mem snd-conv)
qed

lemma turing-machine-enlarged:
  assumes turing-machine k G M and G′ ≥ G
  shows turing-machine k G′ (enlarged G M)
proof
  let ?M = enlarged G M
  show 2 ≤ k and 4 ≤ G′
    using assms turing-machine-def by simp-all
  show turing-command k (length ?M) G′ (?M ! i)
      if i: i < length ?M for i
  proof
    have len: length ?M = length M
      using enlarged-def by simp
    then have 1: turing-command k (length M) G (M ! i)
      using assms(1) that turing-machine-def by simp
    show ⋀gs. length gs = k ⟹ length ([!!] (?M ! i) gs) = length gs
      using enlarged-write that 1 len assms(1) by (metis turing-commandD(1))
    show (?M ! i) gs [.] j < G′
        if length gs = k (⋀i. i < length gs ⟹ gs ! i < G′) j < length gs
        for gs j
    proof (cases symbols-lt G gs)
      case True
      then have (?M ! i) gs = (M ! i) gs
        using enlarged-def i by simp
      moreover have (M ! i) gs [.] j < G
        using 1 turing-commandD(2) that(1,3) True by simp
      ultimately show ?thesis
        using assms(2) by simp
    next
      case False
      then have (?M ! i) gs = (0, map (λr. (r, Stay)) gs)
        using enlarged-def i by auto
      then show ?thesis
        using that by simp
    qed
    show (?M ! i) gs [.] 0 = gs ! 0 if length gs = k and k > 0 for gs
    proof (cases symbols-lt G gs)
      case True
      then show ?thesis
        using enlarged-def i 1 turing-command-def that by simp
    next
      case False
      then have (?M ! i) gs = (0, map (λr. (r, Stay)) gs)
        using that enlarged-def i by auto
      then show ?thesis
        using assms(1) turing-machine-def that by simp
    qed
    show [*] ((?M ! i) gs) ≤ length ?M if length gs = k for gs
    proof (cases symbols-lt G gs)
      case True
      then show ?thesis
        using enlarged-def i that assms(1) turing-machine-def 1 turing-commandD(4) enlarged-nth len
        by (metis (no-types, lifting))
    next
      case False
      then show ?thesis
        using that enlarged-def i by auto
    qed
```

```
  qed
qed
```

The enlarged machine has the same behavior as the original machine when started on symbols over the original alphabet G.

```
lemma execute-enlarged:
  assumes turing-machine k G M and symbols-lt G zs
  shows execute (enlarged G M) (start-config k zs) t = execute M (start-config k zs) t
proof (induction t)
  case 0
  then show ?case
    by simp
next
  case (Suc t)
  let ?M = enlarged G M
  have execute ?M (start-config k zs) (Suc t) = exe ?M (execute ?M (start-config k zs) t)
    by simp
  also have ... = exe ?M (execute M (start-config k zs) t)
      (is - = exe ?M ?cfg)
    using Suc by simp
  also have ... = execute M (start-config k zs) (Suc t)
  proof (cases fst ?cfg < length M)
    case True
    then have exe ?M ?cfg = sem (?M ! (fst ?cfg)) ?cfg
        (is - = sem ?cmd ?cfg)
      using exe-lt-length length-enlarged by simp
    then have exe ?M ?cfg =
        (fst (?cmd (config-read ?cfg)),
         map (λ(a, tp). act a tp) (zip (snd (?cmd (config-read ?cfg))) (snd ?cfg)))
      using sem' by simp
    moreover have symbols-lt G (config-read ?cfg)
      using read-alphabet' assms by auto
    ultimately have exe ?M ?cfg =
        (fst ((M ! (fst ?cfg)) (config-read ?cfg)),
         map (λ(a, tp). act a tp) (zip (snd ((M ! (fst ?cfg)) (config-read ?cfg))) (snd ?cfg)))
      using True enlarged-nth by auto
    then have exe ?M ?cfg = exe M ?cfg
      using sem' by (simp add: True exe-lt-length)
    then show ?thesis
      using Suc by simp
  next
    case False
    then show ?thesis
      using Suc enlarged-def exe-def by auto
  qed
  finally show ?case .
qed

lemma transforms-enlarged:
  assumes turing-machine k G M
    and symbols-lt G zs
    and transforms M (snd (start-config k zs)) t tps1
  shows transforms (enlarged G M) (snd (start-config k zs)) t tps1
proof −
  let ?tps = snd (start-config k zs)
  have ∃t'≤t. execute M (start-config k zs) t' = (length M, tps1)
    using assms(3) transforms-def transits-def start-config-def by simp
  then have ∃t'≤t. execute (enlarged G M) (start-config k zs) t' = (length M, tps1)
    using assms(1,2) transforms-def transits-def execute-enlarged by auto
  moreover have length M = length (enlarged G M)
    using enlarged-def by simp
  ultimately show ?thesis
    using start-config-def transforms-def transitsI by auto
```

qed

2.2.2 Increasing the number of tapes

We can add tapes to a Turing machine in such a way that on the additional tapes the machine does nothing. While the new tapes could go anywhere, we only consider appending them at the end or inserting them at the beginning.

Appending tapes at the end

The next function turns a k-tape Turing machine into a k'-tape Turing machine (for $k' \geq k$) by appending $k' - k$ tapes at the end.

```
definition append-tapes :: nat ⇒ nat ⇒ machine ⇒ machine where
  append-tapes k k' M ≡
    map (λcmd rs. (fst (cmd (take k rs)), snd (cmd (take k rs)) @ (map (λi. (rs ! i, Stay)) [k..<k']))) M

lemma length-append-tapes: length (append-tapes k k' M) = length M
  unfolding append-tapes-def by simp

lemma append-tapes-nth:
  assumes i < length M and length gs = k'
  shows (append-tapes k k' M ! i) gs =
        (fst ((M ! i) (take k gs)), snd ((M ! i) (take k gs)) @ (map (λj. (gs ! j, Stay)) [k..<k']))
  unfolding append-tapes-def using assms(1) by simp

lemma append-tapes-tm:
  assumes turing-machine k G M and k' ≥ k
  shows turing-machine k' G (append-tapes k k' M)
proof
  let ?M = append-tapes k k' M
  show 2 ≤ k'
    using assms turing-machine-def by simp
  show 4 ≤ G
    using assms(1) turing-machine-def by simp
  show turing-command k' (length ?M) G (?M ! i) if i < length ?M for i
  proof
    have i < length M
      using that by (simp add: append-tapes-def)
    then have turing-command: turing-command k (length M) G (M ! i)
      using assms(1) that turing-machine-def by simp
    have ith: append-tapes k k' M ! i =
        (λrs. (fst ((M ! i) (take k rs)), snd ((M ! i) (take k rs)) @ (map (λj. (rs ! j, Stay)) [k..<k'])))
      unfolding append-tapes-def using ‹i < length M› by simp
    show ⋀gs. length gs = k' ⟹ length ([!!] (append-tapes k k' M ! i) gs) = length gs
      using assms(2) ith turing-command turing-commandD by simp
    show (append-tapes k k' M ! i) gs [.] j < G
      if length gs = k' ⋀i. i < length gs ⟹ gs ! i < G j < length gs
      for j gs
    proof (cases j < k)
      case True
      let ?gs = take k gs
      have len: length ?gs = k
        using that(1) assms(2) by simp
      have ⋀i. i < length ?gs ⟹ ?gs ! i < G
        using that(2) by simp
      then have ∀i'<length ?gs. (M ! i) ?gs [.] i' < G
        using turing-commandD(2)[OF turing-command len] by simp
      then show ?thesis
        using ith that turing-commandD(1)[OF turing-command len] by (simp add: nth-append)
    next
      case False
      then have j ≥ k
```

```
        by simp
      have *: length (snd ((M ! i) (take k gs))) = k
        using turing-commandD(1)[OF turing-command] assms(2) that(1) by auto
      have (append-tapes k k' M ! i) gs [.] j =
          fst ((snd ((M ! i) (take k gs)) @ (map (λj. (gs ! j, Stay)) [k..<k'])) ! j)
        using ith by simp
      also have ... = fst ((map (λj. (gs ! j, Stay)) [k..<k']) ! (j − k))
        using * that ‹j ≥ k› by (simp add: False nth-append)
      also have ... = fst (gs ! j, Stay)
        by (metis False ‹k ≤ j› add-diff-inverse-nat diff-less-mono length-upt nth-map nth-upt that(1,3))
      also have ... = gs ! j
        by simp
      also have ... < G
        using that(2,3) by simp
      finally show ?thesis
        by simp
    qed
    show (append-tapes k k' M ! i) gs [.] 0 = gs ! 0 if length gs = k' for gs
    proof −
      have k > 0
        using assms(1) turing-machine-def by simp
      then have 1: (M ! i) rs [.] 0 = rs ! 0 if length rs = k for rs
        using turing-commandD(3)[OF turing-command that] that by simp
      have len: length (take k gs) = k
        by (simp add: assms(2) min-absorb2 that(1))
      then have *: length (snd ((M ! i) (take k gs))) = k
        using turing-commandD(1)[OF turing-command] by auto
      have (append-tapes k k' M ! i) gs [.] 0 =
          fst ((snd ((M ! i) (take k gs)) @ (map (λj. (gs ! j, Stay)) [k..<k'])) ! 0)
        using ith by simp
      also have ... = fst (snd ((M ! i) (take k gs)) ! 0)
        using * by (simp add: nth-append ‹0 < k›)
      finally show ?thesis
        using 1 len ‹0 < k› by simp
    qed
    show [*] ((append-tapes k k' M ! i) gs) ≤ length (append-tapes k k' M) if length gs = k' for gs
    proof −
      have length (take k gs) = k
        using assms(2) that by simp
      then have 1: fst ((M ! i) (take k gs)) ≤ length M
        using turing-commandD[OF turing-command] ‹i < length M› assms(1) turing-machine-def by blast
      moreover have fst ((append-tapes k k' M ! i) gs) = fst ((M ! i) (take k gs))
        using ith by simp
      ultimately show fst ((append-tapes k k' M ! i) gs) ≤ length (append-tapes k k' M)
        using length-append-tapes by metis
    qed
  qed
qed

lemma execute-append-tapes:
  assumes turing-machine k G M and k' ≥ k and length tps = k'
  shows execute (append-tapes k k' M) (q, tps) t =
    (fst (execute M (q, take k tps) t), snd (execute M (q, take k tps) t) @ drop k tps)
proof (induction t)
  case 0
  then show ?case
    by simp
next
  case (Suc t)
  let ?M = append-tapes k k' M
  let ?cfg = execute M (q, take k tps) t
  let ?cfg' = execute M (q, take k tps) (Suc t)
  have execute ?M (q, tps) (Suc t) = exe ?M (execute ?M (q, tps) t)
```

```
    by simp
  also have ... = exe ?M (fst ?cfg, snd ?cfg @ drop k tps)
    using Suc by simp
  also have ... = (fst ?cfg', snd ?cfg' @ drop k tps)
  proof (cases fst ?cfg < length ?M)
    case True
    have sem (?M ! (fst ?cfg)) (fst ?cfg, snd ?cfg @ drop k tps) = (fst ?cfg', snd ?cfg' @ drop k tps)
    proof (rule semI)
      have turing-machine k' G (append-tapes k k' M)
        using append-tapes-tm[OF assms(1,2)] by simp
      then show 1: proper-command k' (append-tapes k k' M ! fst (execute M (q, take k tps) t))
        using True turing-machine-def turing-commandD by (metis nth-mem)
      show 2: length (snd ?cfg @ drop k tps) = k'
        using assms execute-num-tapes by fastforce
      show length (snd ?cfg' @ drop k tps) = k'
        by (metis (no-types, lifting) append-take-drop-id assms execute-num-tapes
          length-append length-take min-absorb2 snd-conv)
      show fst ((?M ! fst ?cfg) (read (snd ?cfg @ drop k tps))) = fst ?cfg'
      proof -
        have less': fst ?cfg < length M
          using True by (simp add: length-append-tapes)
        let ?tps = snd ?cfg @ drop k tps
        have length (snd ?cfg) = k
          using assms execute-num-tapes by fastforce
        then have take2: take k ?tps = snd ?cfg
          by simp
        let ?rs = read ?tps
        have len: length ?rs = k'
          using 2 read-length by simp
        have take2': take k ?rs = read (snd ?cfg)
          using read-def take2 by (metis (mono-tags, lifting) take-map)
        have fst ((?M ! fst ?cfg) ?rs) =
            fst (fst ((M ! fst ?cfg) (take k ?rs)), snd ((M ! fst ?cfg) (take k ?rs)) @ (map (λj. (?rs ! j, Stay))
[k..<k']))
          using append-tapes-nth[OF less' len] by simp
        also have ... = fst ((M ! fst ?cfg) (read (snd ?cfg)))
          using take2' by simp
        also have ... = fst (exe M ?cfg)
          by (simp add: exe-def less' sem-fst)
        finally show ?thesis
          by simp
      qed
      show (act ((?M ! fst ?cfg) (read (snd ?cfg @ drop k tps)) [!] j) ((snd ?cfg @ drop k tps) ! j) =
          (snd ?cfg' @ drop k tps) ! j)
        if j < k' for j
      proof -
        have less': fst ?cfg < length M
          using True by (simp add: length-append-tapes)
        let ?tps = snd ?cfg @ drop k tps
        have len2: length (snd ?cfg) = k
          using assms execute-num-tapes by fastforce
        then have take2: take k ?tps = snd ?cfg
          by simp
        from len2 have len2': length (snd ((M ! fst ?cfg) (read (snd ?cfg)))) = k
          using assms(1) turing-commandD(1) less' read-length turing-machine-def by (metis nth-mem)
        let ?rs = read ?tps
        have len: length ?rs = k'
          using 2 read-length by simp
        have take2': take k ?rs = read (snd ?cfg)
          using read-def take2 by (metis (mono-tags, lifting) take-map)
        have act ((?M ! fst ?cfg) ?rs [!] j) (?tps ! j) =
            act ((fst ((M ! fst ?cfg) (take k ?rs)), snd ((M ! fst ?cfg) (take k ?rs)) @ (map (λj. (?rs ! j, Stay))
[k..<k'])) [!] j) (?tps ! j)
```

using *append-tapes-nth*[*OF less′ len*] **by** *simp*
also have ... = *act*
((*fst* ((*M* ! *fst ?cfg*) (*read* (*snd ?cfg*))), *snd* ((*M* ! *fst ?cfg*) (*read* (*snd ?cfg*))) @ (*map* (λ*j*. (*?rs* ! *j*, *Stay*)) [*k*..<*k′*])) [!] *j*)
(*?tps* ! *j*)
using *take2′* **by** *simp*
also have ... = *act*
((*snd* ((*M* ! *fst ?cfg*) (*read* (*snd ?cfg*))) @ (*map* (λ*j*. (*?rs* ! *j*, *Stay*)) [*k*..<*k′*])) ! *j*)
(*?tps* ! *j*)
by *simp*
also have ... = (*snd ?cfg′* @ *drop k tps*) ! *j*
proof (*cases j* < *k*)
case *True*
then have *tps*: *?tps* ! *j* = *snd ?cfg* ! *j*
by (*simp add*: *len2 nth-append*)
have (*snd ?cfg′* @ *drop k tps*) ! *j* = (*snd* (*exe M ?cfg*) @ *drop k tps*) ! *j*
by *simp*
also have ... = *snd* (*exe M ?cfg*) ! *j*
using *assms*(*1*) *True* **by** (*metis exe-num-tapes len2 nth-append*)
also have ... = *snd* (*sem* (*M* ! *fst ?cfg*) *?cfg*) ! *j*
by (*simp add*: *exe-lt-length less′*)
also have ... = *act* (*snd* ((*M* ! *fst ?cfg*) (*read* (*snd ?cfg*))) ! *j*) (*?tps* ! *j*)
proof −
have *proper-command k* (*M* ! (*fst ?cfg*))
using *turing-commandD*(*1*) *turing-machine-def assms*(*1*) *less′ nth-mem* **by** *blast*
then show *?thesis*
using *sem-snd True tps len2* **by** *simp*
qed
finally show *?thesis*
using *len2′ True* **by** (*simp add*: *nth-append*)
next
case *False*
then have *tps*: *?tps* ! *j* = *tps* ! *j*
using *len2* **by** (*metis* (*no-types*, *lifting*) *2 append-take-drop-id assms*(*3*) *length-take nth-append take2*)
from *False* **have** *gt2*: *j* ≥ *k*
by *simp*
have *len′*: *length* (*snd ?cfg′*) = *k*
using *assms*(*1*) *exe-num-tapes len2* **by** *auto*
have *rs*: *?rs* ! *j* = *read tps* ! *j*
using *tps* **by** (*metis* (*no-types*, *lifting*) *2 assms*(*3*) *that nth-map read-def*)
have *act* ((*snd* ((*M* ! *fst ?cfg*) (*read* (*snd ?cfg*))) @ (*map* (λ*j*. (*?rs* ! *j*, *Stay*)) [*k*..<*k′*])) ! *j*) (*?tps* ! *j*) =
act ((*map* (λ*j*. (*?rs* ! *j*, *Stay*)) [*k*..<*k′*]) ! (*j* − *k*)) (*?tps* ! *j*)
using *False len2 len2′* **by** (*simp add*: *nth-append*)
also have ... = *act* (*?rs* ! *j*, *Stay*) (*?tps* ! *j*)
by (*metis* (*no-types*, *lifting*) *False add-diff-inverse-nat diff-less-mono gt2 that length-upt nth-map nth-upt*)
also have ... = *act* (*?rs* ! *j*, *Stay*) (*tps* ! *j*)
using *tps* **by** *simp*
also have ... = *act* (*read tps* ! *j*, *Stay*) (*tps* ! *j*)
using *rs* **by** *simp*
also have ... = *tps* ! *j*
using *act-Stay assms*(*3*) *that* **by** *simp*
also have ... = (*snd* (*exe M ?cfg*) @ *drop k tps*) ! *j*
using *len′*
by (*metis* (*no-types*, *lifting*) *2 False append-take-drop-id assms*(*3*) *execute.simps*(*2*) *len2 length-take nth-append take2*)
also have ... = (*snd ?cfg′* @ *drop k tps*) ! *j*
by *simp*
finally show *?thesis*
by *simp*
qed
finally show *act* ((*?M* ! *fst ?cfg*) *?rs* [!] *j*) (*?tps* ! *j*) = (*snd ?cfg′* @ *drop k tps*) ! *j* **.**
qed
qed

```
    then show ?thesis
      using exe-def True by simp
  next
    case False
    then show ?thesis
      using assms by (simp add: exe-ge-length length-append-tapes)
  qed
  finally show execute ?M (q, tps) (Suc t) = (fst ?cfg', snd ?cfg' @ drop k tps) .
qed

lemma execute-append-tapes':
  assumes turing-machine k G M and length tps = k
  shows execute (append-tapes k (k + length tps') M) (q, tps @ tps') t =
    (fst (execute M (q, tps) t), snd (execute M (q, tps) t) @ tps')
  using assms execute-append-tapes by simp

lemma transforms-append-tapes:
  assumes turing-machine k G M
    and length tps0 = k
    and transforms M tps0 t tps1
  shows transforms (append-tapes k (k + length tps') M) (tps0 @ tps') t (tps1 @ tps')
    (is transforms ?M - - -)
proof −
  have execute M (0, tps0) t = (length M, tps1)
   using assms(3) transforms-def transits-def by (metis (no-types, opaque-lifting) execute-after-halting-ge fst-conv)
  then have execute ?M (0, tps0 @ tps') t = (length M, tps1 @ tps')
    using assms(1,2) execute-append-tapes' by simp
  moreover have length M = length ?M
    by (simp add: length-append-tapes)
  ultimately show ?thesis
    by (simp add: execute-imp-transits transforms-def)
qed
```

Inserting tapes at the beginning

The next function turns a k-tape Turing machine into a $(k+d)$-tape Turing machine by inserting d tapes at the beginning.

```
definition prepend-tapes :: nat ⇒ machine ⇒ machine where
  prepend-tapes d M ≡
    map (λcmd rs. (fst (cmd (drop d rs)), map (λh. (h, Stay)) (take d rs) @ snd (cmd (drop d rs)))) M

lemma prepend-tapes-at:
  assumes i < length M
  shows (prepend-tapes d M ! i) gs =
    (fst ((M ! i) (drop d gs)), map (λh. (h, Stay)) (take d gs) @ snd ((M ! i) (drop d gs)))
  using assms prepend-tapes-def by simp

lemma prepend-tapes-tm:
  assumes turing-machine k G M
  shows turing-machine (d + k) G (prepend-tapes d M)
proof
  show 2 ≤ d + k
    using assms turing-machine-def by simp
  show 4 ≤ G
    using assms turing-machine-def by simp
  let ?M = prepend-tapes d M
  show turing-command (d + k) (length ?M) G (?M ! i) if i < length ?M for i
  proof
    have len: i < length M
      using that prepend-tapes-def by simp
    then have ∗: (?M ! i) gs = (fst ((M ! i) (drop d gs)), map (λh. (h, Stay)) (take d gs) @ snd ((M ! i) (drop
d gs)))
        if length gs = d + k for gs
```

```
      using prepend-tapes-def that by simp
    have tc: turing-command k (length M) G (M ! i)
      using that turing-machine-def len assms by simp
    show length (snd ((?M ! i) gs)) = length gs if length gs = d + k for gs
      using ∗ that turing-commandD[OF tc] by simp
    show (?M ! i) gs [.] j < G
        if length gs = d + k (⋀i. i < length gs ⟹ gs ! i < G) j < length gs
        for gs j
    proof (cases j < d)
      case True
      have (?M ! i) gs [.] j = fst ((map (λh. (h, Stay)) (take d gs) @ snd ((M ! i) (drop d gs))) ! j)
        using ∗ that(1) by simp
      also have ... = fst (map (λh. (h, Stay)) (take d gs) ! j)
        using True that(1) by (simp add: nth-append)
      also have ... = gs ! j
        by (simp add: True that(3))
      finally have (?M ! i) gs [.] j = gs ! j .
      then show ?thesis
        using that(2,3) by simp
    next
      case False
      have (?M ! i) gs [.] j = fst ((map (λh. (h, Stay)) (take d gs) @ snd ((M ! i) (drop d gs))) ! j)
        using ∗ that(1) by simp
      also have ... = fst (snd ((M ! i) (drop d gs)) ! (j − d))
        using False that(1)
        by (metis (no-types, lifting) add-diff-cancel-left′ append-take-drop-id
          diff-add-inverse2 length-append length-drop length-map nth-append)
      also have ... < G
        using False that turing-commandD[OF tc] by simp
      finally show ?thesis
        by simp
    qed
    show (?M ! i) gs [.] 0 = gs ! 0 if length gs = d + k and d + k > 0 for gs
    proof (cases d = 0)
      case True
      then have (?M ! i) gs [.] 0 = fst (snd ((M ! i) gs) ! 0)
        using ∗ that(1) by simp
      then show ?thesis
        using True that turing-commandD[OF tc] by simp
    next
      case False
      then have (?M ! i) gs [.] 0 = fst ((map (λh. (h, Stay)) (take d gs)) ! 0)
        using ∗ that(1) by (simp add: nth-append)
      also have ... = fst ((map (λh. (h, Stay)) gs) ! 0)
        using False by (metis gr-zeroI nth-take take-map)
      also have ... = gs ! 0
        using False that by simp
      finally show ?thesis
        by simp
    qed
    show [∗] ((?M ! i) gs) ≤ length ?M if length gs = d + k for gs
    proof −
      have fst ((?M ! i) gs) = fst ((M ! i) (drop d gs))
        using that ∗ by simp
      moreover have length (drop d gs) = k
        using that by simp
      ultimately have fst ((?M ! i) gs) ≤ length M
        using turing-commandD(4)[OF tc] by fastforce
      then show fst ((?M ! i) gs) ≤ length ?M
        using prepend-tapes-def by simp
    qed
  qed
qed
```

definition *shift-cfg* :: *tape list* ⇒ *config* ⇒ *config* **where**
shift-cfg tps cfg ≡ (*fst cfg*, *tps* @ *snd cfg*)

lemma *execute-prepend-tapes*:
assumes *turing-machine k G M* **and** *length tps* = *d* **and** ||*cfg0*|| = *k*
shows *execute* (*prepend-tapes d M*) (*shift-cfg tps cfg0*) *t* = *shift-cfg tps* (*execute M cfg0 t*)
proof (*induction t*)
case *0*
show *?case*
by *simp*
next
case (*Suc t*)
let *?M* = *prepend-tapes d M*
let *?scfg* = *shift-cfg tps cfg0*
let *?scfg′* = *execute ?M ?scfg t*
let *?cfg′* = *execute M cfg0 t*
have *fst*: *fst ?cfg′* = *fst ?scfg′*
using *shift-cfg-def Suc.IH* **by** *simp*
have *len*: ||*?cfg′*|| = *k*
using *assms*(*1*,*3*) *execute-num-tapes read-length* **by** *auto*
have *len-s*: ||*?scfg′*|| = *d* + *k*
using *prepend-tapes-tm*[*OF assms*(*1*)] *shift-cfg-def assms*(*2*,*3*) *execute-num-tapes read-length*
by (*metis length-append snd-conv*)
let *?srs* = *read* (*snd ?scfg′*)
let *?rs* = *read* (*snd ?cfg′*)
have *len-rs*: *length ?rs* = *k*
using *assms*(*1*,*3*) *execute-num-tapes read-length* **by** *auto*
moreover have *len-srs*: *length ?srs* = *k* + *d*
using *prepend-tapes-tm*[*OF assms*(*1*)] *shift-cfg-def assms*(*2*,*3*)
by (*metis add.commute execute-num-tapes length-append read-length snd-conv*)
ultimately have *srs-rs*: *drop d ?srs* = *?rs*
using *Suc shift-cfg-def read-def* **by** *simp*
have ∗: *execute ?M ?scfg* (*Suc t*) = *exe ?M ?scfg′*
by *simp*
show *?case*
proof (*cases fst ?scfg′* ≥ *length ?M*)
case *True*
then show *?thesis*
using ∗ *Suc exe-ge-length shift-cfg-def prepend-tapes-def* **by** *auto*
next
case *running*: *False*
then have *scmd*: *?M* ! (*fst ?scfg′*) =
(λ*gs*. (*fst* ((*M* ! (*fst ?scfg′*)) (*drop d gs*)), *map* (λ*h*. (*h*, *Stay*)) (*take d gs*) @ *snd* ((*M* ! (*fst ?scfg′*)) (*drop d gs*))))
(**is** *?scmd* = -)
using *prepend-tapes-at prepend-tapes-def* **by** *auto*
then have *cmd*: *?M* ! (*fst ?scfg′*) =
(λ*gs*. (*fst* ((*M* ! (*fst ?cfg′*)) (*drop d gs*)), *map* (λ*h*. (*h*, *Stay*)) (*take d gs*) @ *snd* ((*M* ! (*fst ?cfg′*)) (*drop d gs*))))
using *fst* **by** *simp*
let *?cmd* = *M* ! (*fst ?cfg′*)

have *execute ?M ?scfg* (*Suc t*) = *sem* (*?M* ! (*fst ?scfg′*)) *?scfg′*
using *running* ∗ *exe-lt-length* **by** *simp*
then have *lhs*: *execute ?M ?scfg* (*Suc t*) =
(*fst* (*?scmd ?srs*), *map* (λ(*a*, *tp*). *act a tp*) (*zip* (*snd* (*?scmd ?srs*)) (*snd ?scfg′*)))
(**is** - = *?lhs*)
using *sem′* **by** *simp*

have *shift-cfg tps* (*execute M cfg0* (*Suc t*)) = *shift-cfg tps* (*exe M ?cfg′*)
by *simp*
also have ... = *shift-cfg tps* (*sem* (*M* ! (*fst ?cfg′*)) *?cfg′*)

```
  using exe-lt-length running fst prepend-tapes-def by auto
also have ... = shift-cfg tps
    (fst (?cmd ?rs), map (λ(a, tp). act a tp) (zip (snd (?cmd ?rs)) (snd ?cfg')))
  using sem' by simp
also have ... = (fst (?cmd ?rs), tps @ map (λ(a, tp). act a tp) (zip (snd (?cmd ?rs)) (snd ?cfg')))
  using shift-cfg-def by simp
finally have rhs: shift-cfg tps (execute M cfg0 (Suc t)) =
   (fst (?cmd ?rs), tps @ map (λ(a, tp). act a tp) (zip (snd (?cmd ?rs)) (snd ?cfg')))
  (is - = ?rhs) .

have ?lhs = ?rhs
proof standard+
  show fst (?scmd ?srs) = fst (?cmd ?rs)
    using srs-rs cmd by simp
  show map (λ(a, tp). act a tp) (zip (snd (?scmd ?srs)) (snd ?scfg')) =
      tps @ map (λ(a, tp). act a tp) (zip (snd (?cmd ?rs)) (snd ?cfg'))
      (is ?l = ?r)
  proof (rule nth-equalityI)
    have lenl: length ?l = d + k
      using lhs execute-num-tapes assms prepend-tapes-tm len-s
      by (smt (verit) length-append shift-cfg-def snd-conv)
    moreover have length ?r = d + k
      using rhs execute-num-tapes assms shift-cfg-def
      by (metis (mono-tags, lifting) length-append snd-conv)
    ultimately show length ?l = length ?r
      by simp
    show ?l ! j = ?r ! j if j < length ?l for j
    proof (cases j < d)
      case True
      let ?at = zip (snd (?scmd ?srs)) (snd ?scfg') ! j
      have ?l ! j = act (fst ?at) (snd ?at)
        using that by simp
      moreover have fst ?at = snd (?scmd ?srs) ! j
        using that by simp
      moreover have snd ?at = snd ?scfg' ! j
        using that by simp
      ultimately have ?l ! j = act (snd (?scmd ?srs) ! j) (snd ?scfg' ! j)
        by simp
      moreover have snd ?scfg' ! j = tps ! j
        using shift-cfg-def assms(2) by (metis (no-types, lifting) Suc.IH True nth-append snd-conv)
      moreover have snd (?scmd ?srs) ! j = (?srs ! j, Stay)
      proof −
        have snd (?scmd ?srs) = map (λh. (h, Stay)) (take d ?srs) @ snd ((M ! (fst ?scfg')) (drop d ?srs))
          using scmd by simp
        then have snd (?scmd ?srs) ! j = map (λh. (h, Stay)) (take d ?srs) ! j
          using len-srs lenl True that
          by (smt (verit) add.commute length-map length-take min-less-iff-conj nth-append)
        then show ?thesis
          using len-srs True by simp
      qed
      moreover have ?r ! j = tps ! j
        using True assms(2) by (simp add: nth-append)
      ultimately show ?thesis
        using len-s that lenl by (metis act-Stay)
    next
      case False
      have jle: j < d + k
        using lenl that by simp
      have jle': j − d < k
        using lenl that False by simp

      let ?at = zip (snd (?scmd ?srs)) (snd ?scfg') ! j
      have ?l ! j = act (fst ?at) (snd ?at)
```

```
          using that by simp
        moreover have fst ?at = snd (?scmd ?srs) ! j
          using that by simp
        moreover have snd ?at = snd ?scfg' ! j
          using that by simp
        ultimately have ?l ! j = act (snd (?scmd ?srs) ! j) (snd ?scfg' ! j)
          by simp
        moreover have snd ?scfg' ! j = snd ?cfg' ! (j − d)
          using shift-cfg-def assms(2) Suc False jle by (metis nth-append snd-conv)
        moreover have snd (?scmd ?srs) ! j = snd (?cmd ?rs) ! (j − d)
        proof −
          have snd (?scmd ?srs) = map (λh. (h, Stay)) (take d ?srs) @ snd ((M ! (fst ?cfg')) (drop d ?srs))
            using cmd by simp
          then have snd (?scmd ?srs) ! j = snd ((M ! (fst ?cfg')) (drop d ?srs)) ! (j − d)
            using len-srs lenl False that len-rs
            by (smt (verit) Nat.add-diff-assoc add.right-neutral add-diff-cancel-left' append-take-drop-id
              le-add1 length-append length-map nth-append srs-rs)
          then have snd (?scmd ?srs) ! j = snd (?cmd ?rs) ! (j − d)
            using srs-rs by simp
          then show ?thesis
            by simp
        qed
        moreover have ?r ! j = act (snd (?cmd ?rs) ! (j − d)) (snd ?cfg' ! (j − d))
        proof −
          have fst (execute M cfg0 t) < length M
            using running fst prepend-tapes-def by simp
          then have len1: length (snd (?cmd ?rs)) = k
            using assms(1) len-rs turing-machine-def[of k G M] turing-commandD(1) by fastforce
          have ?r ! j = map (λ(a, tp). act a tp) (zip (snd (?cmd ?rs)) (snd ?cfg')) ! (j − d)
            using assms(2) False by (simp add: nth-append)
          also have ... = act (snd (?cmd ?rs) ! (j − d)) (snd ?cfg' ! (j − d))
            using len1 len jle' by simp
          finally show ?thesis
            by simp
        qed
        ultimately show ?thesis
          by simp
      qed
    qed
  qed
  then show ?thesis
    using lhs rhs by simp
 qed
qed

lemma transforms-prepend-tapes:
 assumes turing-machine k G M
  and length tps = d
  and length tps0 = k
  and transforms M tps0 t tps1
 shows transforms (prepend-tapes d M) (tps @ tps0) t (tps @ tps1)
proof −
 have ∃t'≤t. execute M (0, tps0) t' = (length M, tps1)
  using assms(4) transforms-def transits-def by simp
 then have ∃t'≤t. execute (prepend-tapes d M) (shift-cfg tps (0, tps0)) t' = shift-cfg tps (length M, tps1)
  using assms transforms-def transits-def execute-prepend-tapes shift-cfg-def by auto
 moreover have length M = length (prepend-tapes d M)
  using prepend-tapes-def by simp
 ultimately show ?thesis
  using shift-cfg-def transforms-def transitsI by auto
qed

end
```

2.3 Combining Turing machines

```
theory Combinations
  imports Basics HOL−Eisbach.Eisbach
begin
```

This section describes how we can combine Turing machines in the way of traditional control structures found in structured programming, namely sequences, branching, and iterating. This allows us to build complex Turing machines from simpler ones and analyze their behavior and running time. Thanks to some syntactic sugar the result may even look like a programming language, but it is really more like a "construction kit" than a "true" programming language with small and big step semantics or Hoare rules. Instead we will merely have some lemmas for proving *transforms* statements for the combined machines. The remaining sections of this chapter will provide us with concrete Turing machines to combine.

2.3.1 Relocated machines

If we use a Turing machine M as part of another TM and there are q commands before M, then M's target states will be off by q. This can be fixed by adding q to all target states of all commands in M, an operation we call *relocation*.

```
definition relocate-cmd :: nat ⇒ command ⇒ command where
  relocate-cmd q cmd rs ≡ (fst (cmd rs) + q, snd (cmd rs))

lemma relocate-cmd-head: relocate-cmd q cmd rs [~] j = cmd rs [~] j
  using relocate-cmd-def by simp

lemma sem-relocate-cmd: sem (relocate-cmd q cmd) cfg = (sem cmd cfg) <+=> q
proof−
  let ?cmd′ = relocate-cmd q cmd
  let ?rs = read (snd cfg)
  have ?cmd′ ?rs = (fst (cmd ?rs) + q, snd (cmd ?rs))
    by (simp add: relocate-cmd-def)
  then show ?thesis
    using sem by (metis (no-types, lifting) fst-conv snd-conv)
qed

definition relocate :: nat ⇒ machine ⇒ machine where
  relocate q M ≡ map (relocate-cmd q) M

lemma relocate:
  assumes M′ = relocate q M and i < length M
  shows (M′ ! i) r = (fst ((M ! i) r) + q, snd ((M ! i) r))
  using assms relocate-def relocate-cmd-def by simp

lemma sem-relocate:
  assumes M′ = relocate q M and i < length M
  shows sem (M′ ! i) cfg = sem (M ! i) cfg <+=> q
  using assms relocate-def sem-relocate-cmd by simp

lemma turing-command-relocate-cmd:
  assumes turing-command k Q G cmd
  shows turing-command k (Q + q) G (relocate-cmd q cmd)
  using assms relocate-cmd-def turing-commandD by auto

lemma turing-command-relocate:
  assumes M′ = relocate q M and turing-machine k G M and i < length M
  shows turing-command k (length M + q) G (M′ ! i)
proof
  fix gs :: symbol list
  assume gs: length gs = k
  have tc: turing-command k (length M) G (M ! i)
    using turing-machine-def assms(2,3) by simp
  show length ([!!] (M′ ! i) gs) = length gs
```

using *gs turing-commandD(1)[OF tc] assms(1,3) relocate* **by** *simp*
show $(M' ! i)$ *gs* [.] *0* = *gs* ! *0* **if** $k > 0$
using *gs turing-commandD(3)[OF tc] assms(1,3) relocate that* **by** *simp*
show [∗] $((M' ! i)$ *gs*$) \leq$ *length M* + *q*
using *gs turing-commandD(4)[OF tc] assms(1,3) relocate* **by** *simp*
show $(\bigwedge i.\ i <$ *length gs* $\Longrightarrow$ *gs* ! $i < G) \Longrightarrow$
$j <$ *length gs* $\Longrightarrow (M' ! i)$ *gs* [.] $j < G$ **for** *j*
using *gs turing-commandD(2)[OF tc] assms(1,3) relocate* **by** *simp*
qed

lemma *length-relocate*: *length* (*relocate q M*) = *length M*
by (*simp add*: *relocate-def*)

lemma *relocate-jump-targets*:
assumes *turing-machine k G M*
and M' = *relocate q M*
and $i <$ *length M*
and *length rs* = *k*
shows *fst* $((M' ! i)$ *rs*$) \leq$ *length M* + *q*
using *turing-machine-def relocate-def assms relocate*
by (*metis turing-commandD(4) diff-add-inverse2 fst-conv le-diff-conv nth-mem*)

lemma *relocate-zero*: *relocate 0 M* = *M*
unfolding *relocate-def relocate-cmd-def* **by** *simp*

2.3.2 Sequences

To execute two Turing machines sequentially we concatenate the two machines, relocating the second one by the length of the first one. In this way the halting state of the first machine becomes the start state of the second machine.

definition *turing-machine-sequential* :: *machine* ⇒ *machine* ⇒ *machine* (**infixl** ‹;;› *55*) **where**
M1 ;; *M2* ≡ *M1* @ (*relocate* (*length M1*) *M2*)

If the number of tapes and the alphabet size match, the concatenation of two Turing machines is again a Turing machine.

lemma *turing-machine-sequential-turing-machine* [*intro*, *simp*]:
assumes *turing-machine k G M1* **and** *turing-machine k G M2*
shows *turing-machine k G* (*M1* ;; *M2*) (**is** *turing-machine k G ?M*)
proof
show *1*: $k \geq 2$
using *assms(1) turing-machine-def* **by** *simp*
show *2*: $G \geq 4$
using *assms(1) turing-machine-def* **by** *simp*
have *len*: *length ?M* = *length M1* + *length M2*
using *relocate-def turing-machine-sequential-def* **by** *simp*
show *3*: *turing-command k* (*length ?M*) *G* (*?M* ! *i*) **if** $i <$ *length ?M* **for** *i*
proof (*cases* $i <$ *length M1*)
case *True*
then show *?thesis*
using *turing-machineD[OF assms(1)] turing-machine-sequential-def len turing-command-mono*
by (*metis* (*no-types*, *lifting*) *le-add1 nth-append*)
next
case *False*
then have $i -$ (*length M1*) $<$ *length M2* (**is** *?i* $<$ *length M2*)
using *False that len* **by** *simp*
then have *turing-command k* (*length ?M*) *G* ((*relocate* (*length M1*) *M2*) ! *?i*)
using *assms(2) turing-command-relocate len* **by** (*simp add*: *add.commute*)
moreover have *?M* ! *i* = (*relocate* (*length M1*) *M2*) ! *?i*
using *False* **by** (*simp add*: *nth-append turing-machine-sequential-def*)
ultimately show *?thesis*
by *simp*
qed

qed

lemma *turing-machine-sequential-empty*: *turing-machine-sequential* [] *M* = *M*
  unfolding *turing-machine-sequential-def* **using** *relocate-zero* **by** *simp*

lemma *turing-machine-sequential-nth*:
  assumes *M* = *M1* ;; *M2* **and** *p* < *length M2*
  shows *M* ! (*p* + *length M1*) = *relocate-cmd* (*length M1*) (*M2* ! *p*)
proof−
  let *?r* = *relocate* (*length M1*) *M2*
  have *M* = *M1* @ *?r*
    using *assms*(*1*) *turing-machine-sequential-def* **by** *simp*
  then have *M* ! (*p* + *length M1*) = *?r* ! *p*
    by (*simp add*: *nth-append*)
  then show *?thesis*
    using *assms*(*2*) *relocate-def* **by** *simp*
qed

lemma *turing-machine-sequential-nth′*:
  assumes *M* = *M1* ;; *M2* **and** *length M1* ≤ *q* **and** *q* < *length M*
  shows *M* ! *q* = *relocate-cmd* (*length M1*) (*M2* ! (*q* − *length M1*))
  using *assms turing-machine-sequential-nth length-relocate turing-machine-sequential-def*
  by (*metis* (*no-types*, *lifting*) *add.assoc diff-add length-append less-add-eq-less*)

lemma *length-turing-machine-sequential*:
  length (*turing-machine-sequential M1 M2*) = *length M1* + *length M2*
  using *turing-machine-sequential-def relocate-def* **by** *simp*

lemma *exe-relocate*:
  exe (*M1* ;; *M2*) (*cfg* <+=> *length M1*) = (*exe M2 cfg*) <+=> *length M1*
  using *sem-relocate-cmd sem-state-indep exe-def turing-machine-sequential-nth length-turing-machine-sequential*
  by (*smt* (*verit*, *ccfv-SIG*) *add.commute diff-add-inverse2 fst-conv le-add2 less-diff-conv2 snd-conv*)

lemma *execute-pre-append*:
  assumes *halts M1 cfg* **and** *fst cfg* = *0* **and** *t* ≤ *running-time M1 cfg*
  shows *execute* ((*M0* ;; *M1*) @ *M2*) (*cfg* <+=> *length M0*) *t* = (*execute M1 cfg t*) <+=> *length M0*
  using *assms*(*3*)
proof (*induction t*)
  case *0*
  then show *?case*
    by *simp*
next
  case (*Suc t*)
  let *?l0* = *length M0*
  let *?M* = (*M0* ;; *M1*) @ *M2*
  let *?cfg-t* = *execute ?M* (*cfg* <+=> *?l0*) *t*
  let *?cfg1-t* = *execute M1 cfg t*
  let *?cmd1* = *M1* ! (*fst ?cfg1-t*)
  let *?cmd* = *?M* ! (*fst ?cfg-t*)
  have ∗: *?cfg1-t* <+=> *?l0* = *?cfg-t*
    using *Suc* **by** *simp*
  then have *fst* (*?cfg1-t* <+=> *?l0*) = *fst ?cfg-t*
    by *simp*
  then have *2*: *fst ?cfg1-t* + *?l0* = *fst ?cfg-t*
    by *simp*
  from ∗ **have** *sndeq*: *snd ?cfg1-t* = *snd ?cfg-t*
    by (*metis snd-conv*)
  have *fst* (*execute M1 cfg t*) ≤ *length M1*
    using *halts-impl-le-length assms*(*1*) *halts-def* **by** *blast*
  moreover have *fst ?cfg1-t* ≠ *length M1*
    using *Suc.prems running-time-def wellorder-Least-lemma*(*2*) **by** *fastforce*
  ultimately have *1*: *fst ?cfg1-t* < *length M1*
    by *simp*

```
  with 2 have relocate-cmd ?l0 ?cmd1 = (M0 ;; M1) ! (fst ?cfg1-t + ?l0)
    by (metis turing-machine-sequential-nth)
  then have relocate-cmd ?l0 ?cmd1 = ?M ! (fst ?cfg1-t + ?l0)
    by (simp add: 1 nth-append length-turing-machine-sequential)
  then have 3: relocate-cmd ?l0 ?cmd1 = ?cmd
    using 2 by simp
  with 1 have execute M1 cfg (Suc t) = sem ?cmd1 ?cfg1-t
    by (simp add: exe-def)
  then have (execute M1 cfg (Suc t)) <+=> ?l0 = (sem ?cmd1 ?cfg1-t) <+=> ?l0
    by simp
  then have (execute M1 cfg (Suc t)) <+=> ?l0 = (sem (relocate-cmd ?l0 ?cmd1) ?cfg1-t)
    using sem-relocate-cmd by simp
  then have rhs: (execute M1 cfg (Suc t)) <+=> ?l0 = sem ?cmd ?cfg1-t
    using 3 by simp
  have execute ?M (cfg <+=> ?l0) (Suc t) = exe ?M ?cfg-t
    by simp
  moreover have fst ?cfg-t < length ?M
    using 1 2 by (simp add: length-turing-machine-sequential)
  ultimately have lhs: execute ?M (cfg <+=> ?l0) (Suc t) = sem ?cmd ?cfg-t
    by (simp add: exe-lt-length)
  have sem ?cmd ?cfg-t = sem ?cmd ?cfg1-t
    using sem-state-indep sndeq by fastforce
  with lhs rhs show ?case
    by simp
qed

lemma transits-pre-append′:
  assumes transforms M1 tps t tps′
  shows transits ((M0 ;; M1) @ M2) (length M0, tps) t (length M0 + length M1, tps′)
proof−
  let ?rt = running-time M1 (0, tps)
  let ?M = (M0 ;; M1) @ M2
  have ?rt ≤ t
    using assms transforms-running-time by simp
  have fst (execute M1 (0, tps) t) = length M1
    using assms(1) execute-after-halting-ge halting-config-def transforms-halting-config transforms-running-time
    by (metis (no-types, opaque-lifting) fst-conv)
  then have ∗: halts M1 (0, tps)
    using halts-def by auto
  have transits M1 (0, tps) t (length M1, tps′)
    using assms(1) transits-def by (simp add: transforms-def)
  then have execute M1 (0, tps) ?rt = (length M1, tps′)
    using assms(1) halting-config-def transforms-halting-config by auto
  moreover have execute ?M (length M0, tps) ?rt = (execute M1 (0, tps) ?rt) <+=> length M0
    using execute-pre-append ∗ by auto
  ultimately have execute ?M (length M0, tps) ?rt = (length M1, tps′) <+=> length M0
    by simp
  then have execute ?M (length M0, tps) ?rt = (length M0 + length M1, tps′)
    by simp
  then show ?thesis
    using ‹?rt ≤ t› transits-def by blast
qed

corollary transits-prepend:
  assumes transforms M1 tps t tps′
  shows transits (M0 ;; M1) (length M0, tps) t (length M0 + length M1, tps′)
  using transits-pre-append′ assms by (metis append-Nil2)

corollary transits-append:
  assumes transforms M1 tps t tps′
  shows transits (M1 @ M2) (0, tps) t (length M1, tps′)
  using transits-pre-append′ turing-machine-sequential-empty assms
  by (metis gen-length-def length-code list.size(3))
```

```
corollary execute-append:
  assumes fst cfg = 0 and halts M1 cfg and t ≤ running-time M1 cfg
  shows execute (M1 @ M2) cfg t = execute M1 cfg t
  using assms execute-pre-append turing-machine-sequential-empty
  by (metis add.right-neutral list.size(3) prod.collapse)

lemma execute-sequential:
  assumes execute M1 cfg1 t1 = cfg1'
    and fst cfg1 = 0
    and t1 = running-time M1 cfg1
    and execute M2 cfg2 t2 = cfg2'
    and cfg1' = cfg2 <+=> length M1
    and halts M1 cfg1
  shows execute (M1 ;; M2) cfg1 (t1 + t2) = cfg2' <+=> length M1
proof−
  let ?M = M1 ;; M2
  have 1: execute ?M cfg1 t1 = cfg1'
    using execute-append assms turing-machine-sequential-def by simp
  then have 2: execute ?M cfg1 (t1 + t2) = execute ?M cfg1' t2
    using execute-additive by simp
  have execute M2 cfg2 t2 = cfg2' ⟹ execute ?M cfg1' t2 = cfg2' <+=> length M1 for t2
  proof (induction t2 arbitrary: cfg2')
    case 0
    then show ?case
      using 1 assms(5) by simp
  next
    case (Suc t2)
    let ?cfg = execute M2 cfg2 t2
    have execute ?M cfg1' (Suc t2) = exe ?M (execute ?M cfg1' t2)
      by simp
    then have execute ?M cfg1' (Suc t2) = exe ?M (?cfg <+=> length M1)
      using Suc by simp
    moreover have execute M2 cfg2 (Suc t2) = exe M2 ?cfg
      by simp
    ultimately show ?case
      using exe-relocate Suc.prems by metis
  qed
  then show ?thesis
    using assms(4) 2 by simp
qed
```

The semantics and running time of the ;; operator:

```
lemma transforms-turing-machine-sequential:
  assumes transforms M1 tps1 t1 tps2 and transforms M2 tps2 t2 tps3
  shows transforms (M1 ;; M2) tps1 (t1 + t2) tps3
proof−
  let ?M = M1 ;; M2
  let ?cfg1 = (0, tps1)
  let ?cfg1' = (length M1, tps2)
  let ?t1 = running-time M1 ?cfg1
  let ?cfg2 = (0, tps2)
  let ?cfg2' = (length M2, tps3)
  let ?t2 = running-time M2 ?cfg2
  have fst (execute M1 ?cfg1 ?t1) = length M1
    using assms(1) halting-config-def transforms-halting-config by (metis fst-conv)
  then have ∗: halts M1 ?cfg1
    using halts-def by auto
  have execute M1 ?cfg1 ?t1 = ?cfg1'
    using assms(1) halting-config-def transforms-halting-config by auto
  moreover have fst ?cfg1 = 0
    by simp
  moreover have execute M2 ?cfg2 ?t2 = ?cfg2'
```

using *assms*(*2*) *halting-config-def transforms-halting-config* **by** *auto*
moreover have *?cfg1′* = *?cfg2* <+=> *length M1*
by *simp*
ultimately have *execute ?M ?cfg1* (*?t1* + *?t2*) = *?cfg2′* <+=> *length M1*
using *execute-sequential* ∗ **by** *blast*
then have *execute ?M ?cfg1* (*?t1* + *?t2*) = (*length ?M*, *tps3*)
by (*simp add*: *length-turing-machine-sequential*)
then have *transits ?M ?cfg1* (*?t1* + *?t2*) (*length ?M*, *tps3*)
using *transits-def* **by** *blast*
moreover have *?t1* + *?t2* ≤ *t1* + *t2*
using *add-le-mono assms*(*1*,*2*) *transforms-running-time* **by** *blast*
ultimately have *transits ?M ?cfg1* (*t1* + *t2*) (*length ?M*, *tps3*)
using *transits-monotone* **by** *simp*
then show *?thesis*
using *transforms-def* **by** *simp*
qed

corollary *transforms-tm-sequentialI*:
assumes *transforms M1 tps1 t1 tps2* **and** *transforms M2 tps2 t2 tps3* **and** *t12* = *t1* + *t2*
shows *transforms* (*M1* ;; *M2*) *tps1 t12 tps3*
using *assms transforms-turing-machine-sequential* **by** *simp*

2.3.3 Branches

A branching Turing machine consists of a condition and two Turing machines, one for each of the branches. The condition can be any predicate over the list of symbols read from the tapes. The branching TM thus needs to perform conditional jumps, for which we will have the following command:

definition *cmd-jump* :: (*symbol list* ⇒ *bool*) ⇒ *state* ⇒ *state* ⇒ *command* **where**
cmd-jump cond q1 q2 rs ≡ (*if cond rs then q1 else q2*, *map* (*λr.* (*r*, *Stay*)) *rs*)

lemma *turing-command-jump-1*:
assumes *q1* ≤ *q2* **and** *k* > *0*
shows *turing-command k q2 G* (*cmd-jump cond q1 q2*)
using *assms cmd-jump-def turing-commandI* **by** *simp*

lemma *turing-command-jump-2*:
assumes *q2* ≤ *q1* **and** *k* > *0*
shows *turing-command k q1 G* (*cmd-jump cond q1 q2*)
using *assms cmd-jump-def turing-commandI* **by** *simp*

lemma *sem-jump-snd*: *snd* (*sem* (*cmd-jump cond q1 q2*) *cfg*) = *snd cfg*
proof−
let *?k* = ||*cfg*||
let *?cmd* = *cmd-jump cond q1 q2*
let *?cfg′* = *sem ?cmd cfg*
let *?rs* = *read* (*snd cfg*)
have *1*: *proper-command ?k ?cmd*
using *cmd-jump-def* **by** *simp*
then have *len*: ||*?cfg′*|| = ||*cfg*||
using *sem-num-tapes-raw* **by** *simp*
have *snd ?cfg′* ! *i* = *act* (*snd* (*?cmd ?rs*) ! *i*) (*snd cfg* ! *i*) **if** *i* < ||*cfg*|| **for** *i*
using *sem-snd 1 that* **by** *simp*
moreover have *snd* (*?cmd ?rs*) ! *i* = (*?rs*!*i*, *Stay*) **if** *i* < ||*cfg*|| **for** *i*
using *cmd-jump-def* **by** (*simp add*: *read-length that*)
ultimately have *snd ?cfg′* ! *i* = *act* (*?rs* ! *i*, *Stay*) (*snd cfg* ! *i*) **if** *i* < ||*cfg*|| **for** *i*
using *that* **by** *simp*
then have *snd ?cfg′* ! *i* = *snd cfg* ! *i* **if** *i* < ||*cfg*|| **for** *i*
using *that act-Stay* **by** *simp*
then show *snd ?cfg′* = *snd cfg*
using *len nth-equalityI* **by** *force*
qed

lemma *sem-jump-fst1*:

assumes *cond (read (snd cfg))*
shows *fst (sem (cmd-jump cond q1 q2) cfg) = q1*
using *cmd-jump-def sem assms* **by** *simp*

lemma *sem-jump-fst2*:
assumes ¬ *cond (read (snd cfg))*
shows *fst (sem (cmd-jump cond q1 q2) cfg) = q2*
using *cmd-jump-def sem assms* **by** *simp*

lemma *sem-jump*:
sem (cmd-jump cond q1 q2) cfg = (if cond (config-read cfg) then q1 else q2, snd cfg)
using *sem-def sem-jump-snd cmd-jump-def* **by** *simp*

lemma *transits-jump*:
transits [cmd-jump cond q1 q2] (0, tps) 1 (if cond (read tps) then q1 else q2, tps)
using *transits-def sem-jump exe-def* **by** *auto*

definition *turing-machine-branch* :: *(symbol list ⇒ bool) ⇒ machine ⇒ machine ⇒ machine*
(‹IF - THEN - ELSE - ENDIF› 60)
where
IF cond THEN M1 ELSE M2 ENDIF ≡
[cmd-jump cond 1 (length M1 + 2)] @
(relocate 1 M1) @
[cmd-jump (λ-. True) (length M1 + length M2 + 2) 0] @
(relocate (length M1 + 2) M2)

lemma *turing-machine-branch-len*:
length (IF cond THEN M1 ELSE M2 ENDIF) = length M1 + length M2 + 2
unfolding *turing-machine-branch-def* **by** *(simp add: relocate-def)*

If the Turing machines for both branches have the same number of tapes and the same alphabet size, the branching machine is a Turing machine, too.

lemma *turing-machine-branch-turing-machine*:
assumes *turing-machine k G M1* **and** *turing-machine k G M2*
shows *turing-machine k G (IF cond THEN M1 ELSE M2 ENDIF)*
(**is** *turing-machine - - ?M*)
proof
show *k ≥ 2*
using *assms(2) turing-machine-def* **by** *simp*
show *G ≥ 4*
using *assms(2) turing-machine-def* **by** *simp*
let *?C1 = [cmd-jump cond 1 (length M1 + 2)]*
let *?C2 = relocate 1 M1*
let *?C3 = [cmd-jump (λ-. True) (length M1 + length M2 + 2) 0]*
let *?C4 = relocate (length M1 + 2) M2*
have *parts*: *?M = ?C1 @ ?C2 @ ?C3 @ ?C4*
using *turing-machine-branch-def* **by** *simp*
have *len*: *length ?M = length M1 + length M2 + 2*
using *turing-machine-branch-len* **by** *simp*
have *k > 0*
using *‹k ≥ 2›* **by** *simp*
show *turing-command k (length ?M) G (?M ! i)* **if** *i < length ?M* **for** *i*
proof −
consider
i < length ?C1
| *length ?C1 ≤ i ∧ i < length (?C1 @ ?C2)*
| *length (?C1 @ ?C2) ≤ i ∧ i < length (?C1 @ ?C2 @ ?C3)*
| *length (?C1 @ ?C2 @ ?C3) ≤ i ∧ i < length ?M*
using *‹i < length ?M›* **by** *linarith*
then show *?thesis*
proof *(cases)*
case *1*
then have *turing-command k (length M1 + 2) G (?C1 ! i)*

using *turing-command-jump-1* ‹*k* > *0*› **by** *simp*
then have *turing-command k* (*length ?M*) *G* (*?C1* ! *i*)
using *turing-command-mono len* **by** *simp*
then show *?thesis*
using *parts 1* **by** *simp*
next
case *2*
then have *i* − *length ?C1* < *length ?C2*
by *auto*
then have *turing-command k* (*length M1* + *1*) *G* (*?C2* ! (*i* − *length ?C1*))
using *turing-command-relocate assms length-relocate* **by** *metis*
then have *turing-command k* (*length ?M*) *G* (*?C2* ! (*i* − *length ?C1*))
using *turing-command-mono len* **by** *simp*
then have *turing-command k* (*length ?M*) *G* ((*?C1* @ *?C2*) ! *i*)
using *2* **by** *simp*
then show *?thesis*
using *parts 2* **by** (*metis* (*no-types*, *lifting*) *append.assoc nth-append*)
next
case *3*
then have *turing-command k* (*length ?M*) *G* (*?C3* ! (*i* − *length* (*?C1* @ *?C2*)))
using *turing-command-jump-2 len* ‹*k* > *0*› **by** *simp*
then have *turing-command k* (*length ?M*) *G* ((*?C1* @ *?C2* @ *?C3*) ! *i*)
using *3* **by** (*metis* (*no-types*, *lifting*) *append.assoc diff-is-0-eq′ less-numeral-extra*(*3*) *nth-append zero-less-diff*)
then show *?thesis*
using *parts 3* **by** (*smt* (*verit*, *best*) *append.assoc nth-append*)
next
case *4*
then have *i* − *length* (*?C1* @ *?C2* @ *?C3*) < *length ?C4*
by (*simp add*: *len less-diff-conv2 length-relocate*)
then have *turing-command k* (*length ?M*) *G* (*?C4* ! (*i* − *length* (*?C1* @ *?C2* @ *?C3*)))
using *turing-command-relocate assms* **by** (*smt* (*verit*, *ccfv-SIG*) *add.assoc add.commute len length-relocate*)
then show *?thesis*
using *parts 4* **by** (*metis* (*no-types*, *lifting*) *append.assoc le-simps*(*3*) *not-less-eq-eq nth-append*)
qed
qed
qed

If the condition is true, the branching TM executes M_1 and requires two extra steps: one for evaluating the condition and one for the unconditional jump to the halting state.

lemma *transforms-branch-true*:
assumes *transforms M1 tps t tps′* **and** *cond* (*read tps*)
shows *transforms* (*IF cond THEN M1 ELSE M2 ENDIF*) *tps* (*t* + *2*) *tps′*
(**is** *transforms ?M* - - -)
proof−
let *?C1* = [*cmd-jump cond 1* (*length M1* + *2*)]
let *?C2* = *relocate 1 M1*
let *?C3* = [*cmd-jump* (λ-. *True*) (*length M1* + *length M2* + *2*) *0*]
let *?C4* = *relocate* (*length M1* + *2*) *M2*
let *?C34* = *?C3* @ *?C4*
have *parts*: *?M* = *?C1* @ *?C2* @ *?C3* @ *?C4*
using *turing-machine-branch-def* **by** *simp*
then have *?M* = *?C1* @ *?C2* @ *?C34*
by *simp*
moreover have *?C1* @ *?C2* = *?C1* ;; *M1*
using *turing-machine-sequential-def* **by** *simp*
ultimately have *parts2*: *?M* = (*?C1* ;; *M1*) @ *?C34*
by *simp*
have *execute ?M* (*0*, *tps*) *1* = *exe ?M* (*0*, *tps*)
by *simp*
also have ... = *sem* (*?M* ! *0*) (*0*, *tps*)
using *exe-def* **by** (*simp add*: *turing-machine-branch-len*)
also have ... = *sem* (*?C1* ! *0*) (*0*, *tps*)
by (*simp add*: *parts*)

also have ... = *sem* (*cmd-jump cond 1* (*length M1* + *2*)) (*0*, *tps*)
  by *simp*
also have ... = (*1*, *tps*)
  using *sem-jump* **by** (*simp add*: *assms*(*2*))
finally have *execute ?M* (*0*, *tps*) *1* = (*1*, *tps*) **.**
then have *phase1*: *transits ?M* (*0*, *tps*) *1* (*1*, *tps*)
  using *transits-def* **by** *auto*
have *length ?C1* = *1*
  by *simp*
moreover have *transits* ((*?C1* ;; *M1*) @ *?C34*) (*length ?C1*, *tps*) *t* (*length ?C1* + *length M1*, *tps′*)
  using *transits-pre-append′ assms* **by** *blast*
ultimately have *transits ?M* (*1*, *tps*) *t* (*1* + *length M1*, *tps′*)
  using *parts2* **by** *simp*
with *phase1* **have** *transits ?M* (*0*, *tps*) (*t* + *1*) (*1* + *length M1*, *tps′*)
  using *transits-additive* **by** *fastforce*
then have *phase2*: *transits ?M* (*0*, *tps*) (*t* + *1*) (*length* (*?C1* @ *?C2*), *tps′*)
  by (*simp add*: *relocate-def*)
let *?cfg* = (*length* (*?C1* @ *?C2*), *tps′*)
have ∗: *?M* ! (*length* (*?C1* @ *?C2*)) = *?C3* ! *0*
  using *parts* **by** *simp*
then have *execute ?M ?cfg 1* = *exe ?M ?cfg*
  by *simp*
also have ... = *sem* (*cmd-jump* (λ-. *True*) (*length M1* + *length M2* + *2*) *0*) *?cfg*
  using *exe-def relocate-def turing-machine-branch-len* ∗ **by** (*simp add*: *Suc-le-lessD*)
also have ... = (*length M1* + *length M2* + *2*, *snd ?cfg*)
  using *sem-jump* **by** *simp*
also have ... = (*length ?M*, *tps′*)
  by (*simp add*: *turing-machine-branch-len*)
finally have *execute ?M ?cfg 1* = (*length ?M*, *tps′*) **.**
then have *transits ?M ?cfg 1* (*length ?M*, *tps′*)
  using *transits-def* **by** *auto*
with *phase2* **have** *transits ?M* (*0*, *tps*) (*t* + *2*) (*length ?M*, *tps′*)
  using *transits-additive* **by** *fastforce*
then show *?thesis*
  by (*simp add*: *transforms-def*)
qed

If the condition is false, the branching TM executes M_2 and requires one extra step to evaluate the condition.

lemma *transforms-branch-false*:
  assumes *transforms M2 tps t tps′* **and** ¬ *cond* (*read tps*)
  shows *transforms* (*IF cond THEN M1 ELSE M2 ENDIF*) *tps* (*t* + *1*) *tps′*
  (**is** *transforms ?M* - - -)
proof−
  let *?C1* = [*cmd-jump cond 1* (*length M1* + *2*)]
  let *?C2* = *relocate 1 M1*
  let *?C3* = [*cmd-jump* (λ-. *True*) (*length M1* + *length M2* + *2*) *0*]
  let *?C4* = *relocate* (*length M1* + *2*) *M2*
  let *?C123* = *?C1* @ *?C2* @ *?C3*
  have *parts*: *?M* = *?C1* @ *?C2* @ *?C3* @ *?C4*
    using *turing-machine-branch-def* **by** *simp*
  moreover have *len123*: *length ?C123* = *length M1* + *2*
    by (*simp add*: *length-relocate*)
  ultimately have *seq*: *?M* = *?C123* ;; *M2*
    by (*simp add*: *turing-machine-sequential-def*)
  have *execute ?M* (*0*, *tps*) *1* = *exe ?M* (*0*, *tps*)
    by *simp*
  also have ... = *sem* (*?M* ! *0*) (*0*, *tps*)
    using *exe-def* **by** (*simp add*: *turing-machine-branch-len*)
  also have ... = *sem* (*cmd-jump cond 1* (*length M1* + *2*)) (*0*, *tps*)
    by (*simp add*: *parts*)
  also have ... = (*length M1* + *2*, *tps*)
    using *assms*(*2*) *sem-jump* **by** *simp*

```
  also have ... = (length ?C123, tps)
    using len123 by simp
  finally have execute ?M (0, tps) 1 = (length ?C123, tps) .
  then have phase1: transits ?M (0, tps) 1 (length ?C123, tps)
    using transits-def by blast
  have ?M ! (length ?C123) = ?C4 ! 0
    by (simp add: nth-append parts length-relocate)
  have transits (?C123 ;; M2) (length ?C123, tps) t (length ?C123 + length M2, tps')
    using transits-prepend assms by blast
  then have transits ?M (length ?C123, tps) t (length ?M, tps')
    by (simp add: seq length-turing-machine-sequential)
  with phase1 have transits ?M (0, tps) (t + 1) (length ?M, tps')
    using transits-additive by fastforce
  then show ?thesis
    using transforms-def by simp
qed
```

The behavior and running time of the branching Turing machine:

```
lemma transforms-branch-full:
  assumes c ⟹ transforms M1 tps tT tpsT
    and ¬ c ⟹ transforms M2 tps tF tpsF
    and c ⟹ tT + 2 ≤ t
    and ¬ c ⟹ tF + 1 ≤ t
    and c = cond (read tps)
    and tps' = (if c then tpsT else tpsF)
  shows transforms (IF cond THEN M1 ELSE M2 ENDIF) tps t tps'
proof −
  have transforms (IF cond THEN M1 ELSE M2 ENDIF)
      tps
      (if c then tT + 2 else tF + 1)
      (if c then tpsT else tpsF)
    using assms(1,2,5) transforms-branch-true transforms-branch-false by simp
  moreover have (if c then tT + 2 else tF + 1) ≤ t
    using assms(3,4) by simp
  ultimately show ?thesis
    using transforms-monotone assms(6) by presburger
qed

corollary transforms-branchI:
  assumes cond (read tps) ⟹ transforms M1 tps tT tpsT
    and ¬ cond (read tps) ⟹ transforms M2 tps tF tpsF
    and cond (read tps) ⟹ tT + 2 ≤ t
    and ¬ cond (read tps) ⟹ tF + 1 ≤ t
    and cond (read tps) ⟹ tps' = tpsT
    and ¬ cond (read tps) ⟹ tps' = tpsF
  shows transforms (IF cond THEN M1 ELSE M2 ENDIF) tps t tps'
  by (rule transforms-branch-full) (use assms in auto)
```

2.3.4 Loops

The loops are while loops consisting of a head and a body. The body is a Turing machine that is executed in every iteration as long as the condition in the head of the loop evaluates to true. The condition is of the same form as in branching TMs, namely a predicate over the symbols read from the tapes. Sometimes this is not expressive enough, and so we allow a Turing machine as part of the loop head that is run prior to evaluating the condition. In most cases, however, this TM is empty.

```
definition turing-machine-loop :: machine ⇒ (symbol list ⇒ bool) ⇒ machine ⇒ machine
  (‹WHILE - ; - DO - DONE› 60)
where
  WHILE M1 ; cond DO M2 DONE ≡
    M1 @
    [cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)] @
    (relocate (length M1 + 1) M2) @
```

[*cmd-jump* (λ-. *True*) *0 0*]

Intuitively the Turing machine *WHILE M1 ; cond DO M2 DONE* first executes *M1* and then checks the condition *cond*. If it is true, it executes *M2* and jumps back to the start state; otherwise it jumps to the halting state.

```
lemma turing-machine-loop-len:
  length (WHILE M1 ; cond DO M2 DONE) = length M1 + length M2 + 2
  unfolding turing-machine-loop-def by (simp add: relocate-def)
```

If both Turing machines have the same number of tapes and alphabet size, then so does the looping Turing machine.

```
lemma turing-machine-loop-turing-machine:
  assumes turing-machine k G M1 and turing-machine k G M2
  shows turing-machine k G (WHILE M1 ; cond DO M2 DONE)
    (is turing-machine - - ?M)
proof
  show 1: k ≥ 2
    using assms(1) turing-machine-def by simp
  show 2: G ≥ 4
    using assms(1) turing-machine-def by simp
  let ?C1 = M1
  let ?C2 = [cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)]
  let ?C3 = relocate (length M1 + 1) M2
  let ?C4 = [cmd-jump (λ-. True) 0 0]
  let ?C34 = ?C3 @ ?C4
  have parts: ?M = ?C1 @ ?C2 @ ?C3 @ ?C4
    using turing-machine-loop-def by simp
  have len: length ?M = length M1 + length M2 + 2
    using turing-machine-loop-len by simp
  have k > 0
    using ‹k ≥ 2› by simp

  show turing-command k (length ?M) G (?M ! i) if i < length ?M for i
  proof −
    consider
        i < length ?C1
      | length ?C1 ≤ i ∧ i < length (?C1 @ ?C2)
      | length (?C1 @ ?C2) ≤ i ∧ i < length (?C1 @ ?C2 @ ?C3)
      | length (?C1 @ ?C2 @ ?C3) ≤ i ∧ i < length ?M
      using ‹i < length ?M› by linarith
    then show ?thesis
    proof (cases)
      case 1
      then have turing-command k (length M1) G (?C1 ! i)
        using turing-machineD(3) assms by simp
      then have turing-command k (length ?M) G (?C1 ! i)
        using turing-command-mono len by simp
      then show ?thesis
        using parts 1 by (simp add: nth-append)
    next
      case 2
      then have turing-command k (length M1 + length M2 + 2) G (?C2 ! (i − length ?C1))
        using turing-command-jump-1 ‹0 < k› by simp
      then have turing-command k (length ?M) G (?C2 ! (i − length ?C1))
        using len by simp
      then have turing-command k (length ?M) G ((?C1 @ ?C2) ! i)
        using 2 le-add-diff-inverse by (simp add: nth-append)
      then show ?thesis
        using 2 parts by (simp add: nth-append)
    next
      case 3
      then have turing-command k (length M2 + (length M1 + 1)) G (?C3 ! (i − length (?C1 @ ?C2)))
        using turing-command-relocate length-relocate assms(2)
```

```
      by (smt (verit, best) add.commute add.left-commute add-less-cancel-left le-add-diff-inverse length-append)
    then have turing-command k (length ?M) G (?C3 ! (i − length (?C1 @ ?C2)))
      using turing-command-mono len by simp
    then have turing-command k (length ?M) G ((?C1 @ ?C2 @ ?C3) ! i)
      using 3 by (simp add: nth-append)
    then show ?thesis
      using parts 3 by (smt (verit) append.assoc nth-append)
  next
    case 4
    then have turing-command k 0 G (?C4 ! (i − length (?C1 @ ?C2 @ ?C3)))
      using turing-command-jump-1 ‹0 < k› len length-relocate by simp
    then have turing-command k (length ?M) G (?C4 ! (i − length (?C1 @ ?C2 @ ?C3)))
      using turing-command-mono by blast
    then show ?thesis
      using parts 4 by (metis (no-types, lifting) append.assoc nth-append verit-comp-simplify1(3))
  qed
 qed
qed

lemma transits-turing-machine-loop-cond-false:
 assumes transforms M1 tps t1 tps' and ¬ cond (read tps')
 shows transits
        (WHILE M1 ; cond DO M2 DONE)
        (0, tps)
        (t1 + 1)
        (length M1 + length M2 + 2, tps')
  (is transits ?M - - -)
proof−
 let ?C1 = M1
 let ?C2 = [cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)]
 let ?C3 = relocate (length M1 + 1) M2
 let ?C4 = [cmd-jump (λ-. True) 0 0]
 let ?C34 = ?C3 @ ?C4
 have parts: ?M = ?C1 @ ?C2 @ ?C3 @ ?C4
  using turing-machine-loop-def by simp
 then have ?M = ?C1 @ (?C2 @ ?C3 @ ?C4)
  by simp
 then have transits ?M (0, tps) t1 (length ?C1, tps')
  using assms transits-append by simp
 moreover have transits ?M (length M1, tps') 1 (length M1 + length M2 + 2, tps')
 proof−
  have ∗: ?M ! (length ?C1) = cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)
    using turing-machine-loop-def by simp
  have execute ?M (length ?C1, tps') 1 = exe ?M (length ?C1, tps')
    by simp
  also have ... = sem (?M ! (length ?C1)) (length ?C1, tps')
    by (simp add: exe-lt-length turing-machine-loop-len)
  also have ... = sem (cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)) (length ?C1, tps')
    using ∗ by simp
  also have ... = (length M1 + length M2 + 2, tps')
    using sem-jump assms(2) by simp
  finally have execute ?M (length ?C1, tps') 1 = (length M1 + length M2 + 2, tps') .
  then show ?thesis
    using transits-def by auto
 qed
 ultimately show transits ?M (0, tps) (t1 + 1) (length M1 + length M2 + 2, tps')
  using transits-additive by blast
qed

lemma transits-turing-machine-loop-cond-true:
 assumes transforms M1 tps t1 tps'
  and transforms M2 tps' t2 tps''
  and cond (read tps')
```

```
  shows transits
          (WHILE M1 ; cond DO M2 DONE)
          (0, tps)
          (t1 + t2 + 2)
          (0, tps'')
    (is transits ?M - - -)
proof−
  let ?C1 = M1
  let ?C2 = [cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)]
  let ?C3 = relocate (length M1 + 1) M2
  let ?C4 = [cmd-jump (λ-. True) 0 0]
  let ?C34 = ?C3 @ ?C4
  have parts: ?M = ?C1 @ ?C2 @ ?C3 @ ?C4
    using turing-machine-loop-def by simp
  then have ?M = ?C1 @ (?C2 @ ?C3 @ ?C4)
    by simp
  then have transits ?M (0, tps) t1 (length ?C1, tps')
    using assms(1,3) transits-append by simp
  moreover have transits ?M (length ?C1, tps') 1 (length ?C1 + 1, tps')
  proof−
    have ∗: ?M ! (length ?C1) = cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)
      using turing-machine-loop-def by simp
    have execute ?M (length ?C1, tps') 1 = exe ?M (length ?C1, tps')
      by simp
    also have ... = sem (?M ! (length ?C1)) (length ?C1, tps')
      by (simp add: exe-lt-length turing-machine-loop-len)
    also have ... = sem (cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)) (length ?C1, tps')
      using ∗ by simp
    also have ... = (length M1 + 1, tps')
      using sem-jump assms(3) by simp
    finally have execute ?M (length ?C1, tps') 1 = (length M1 + 1, tps') .
    then show ?thesis
      using transits-def by auto
  qed
  ultimately have transits ?M (0, tps) (t1 + 1) (length M1 + 1, tps')
    using transits-additive by blast
  moreover have transits ?M (length M1 + 1, tps') t2 (length M1 + length M2 + 1, tps'')
  proof−
    have ?M = ((?C1 @ ?C2) ;; M2) @ ?C4
      by (simp add: parts turing-machine-sequential-def)
    moreover have length (?C1 @ ?C2) = length M1 + 1
      by simp
    ultimately have transits ?M (length M1 + 1, tps') t2 (length M1 + 1 + length M2, tps'')
      using assms transits-pre-append' by metis
    then show ?thesis
      by simp
  qed
  ultimately have transits ?M (0, tps) (t1 + t2 + 1) (length M1 + length M2 + 1, tps'')
    using transits-additive by fastforce
  moreover have transits ?M (length M1 + length M2 + 1, tps'') 1 (0, tps'')
  proof−
    have ∗: ?M ! (length M1 + length M2 + 1) = cmd-jump (λ-. True) 0 0
      by (simp add: nth-append parts length-relocate)
    have execute ?M (length M1 + length M2 + 1, tps'') 1 = exe ?M (length M1 + length M2 + 1, tps'')
      by simp
    also have ... = sem (?M ! (length M1 + length M2 + 1)) (length M1 + length M2 + 1, tps'')
      by (simp add: exe-lt-length turing-machine-loop-len)
    also have ... = sem (cmd-jump (λ-. True) 0 0) (length M1 + length M2 + 1, tps'')
      using ∗ by simp
    also have ... = (0, tps'')
      using sem-jump by simp
    finally have execute ?M (length M1 + length M2 + 1, tps'') 1 = (0, tps'') .
    then show ?thesis
```

using *transits-def* **by** *auto*
qed
ultimately show *transits ?M* $(0, tps)$ $(t1 + t2 + 2)$ $(0, tps'')$
using *transits-additive* **by** *fastforce*
qed

In this article we will only encounter while loops that are in fact for loops, that is, where the number of iterations is known beforehand. Moreover, using the same time bound for every iteration will lead to a good enough upper bound for the entire loop.

The *transforms* rule for a loop with m iterations where the running time of both TMs is bounded by a constant:

lemma *transforms-loop-for*:
fixes *m t1 t2* :: *nat*
and *M1 M2* :: *machine*
and *tps* :: *nat* ⇒ *tape list*
and *tps'* :: *nat* ⇒ *tape list*
assumes $\bigwedge i.\ i \le m \Longrightarrow$ *transforms M1* (*tps i*) *t1* (*tps' i*)
assumes $\bigwedge i.\ i < m \Longrightarrow$ *transforms M2* (*tps' i*) *t2* (*tps* (*Suc i*))
and $\bigwedge i.\ i < m \Longrightarrow$ *cond* (*read* (*tps' i*))
and ¬ *cond* (*read* (*tps' m*))
assumes $ttt \ge m * (t1 + t2 + 2) + t1 + 1$
shows *transforms*
(*WHILE M1* ; *cond DO M2 DONE*)
(*tps 0*)
ttt
(*tps' m*)
proof −
define *M* **where** *M* = *WHILE M1* ; *cond DO M2 DONE*
define *t* **where** $t = t1 + t2 + 2$
have *transits M* (*0*, *tps 0*) $(i * t)$ (*0*, *tps i*) **if** $i \le m$ **for** *i*
using *that*
proof (*induction i*)
case *0*
then show *?case*
using *transits-def* **by** *simp*
next
case (*Suc i*)
then have $i < m$
by *simp*
then have *transits M* (*0*, *tps i*) *t* (*0*, *tps* (*Suc i*))
using *M-def t-def assms transits-turing-machine-loop-cond-true* **by** (*metis le-eq-less-or-eq*)
moreover have *transits M* (*0*, *tps 0*) $(i * t)$ (*0*, *tps i*)
using *Suc* **by** *simp*
ultimately have *transits M* (*0*, *tps 0*) $(i * t + t)$ (*0*, *tps* (*Suc i*))
using *transits-additive* **by** *simp*
then show *?case*
by (*metis add.commute mult-Suc*)
qed
then have *transits M* (*0*, *tps 0*) $(m * t)$ (*0*, *tps m*)
by *simp*
moreover have *transits M* (*0*, *tps m*) $(t1 + 1)$ (*length M1* + *length M2* + *2*, *tps' m*)
using *assms*(*1*,*4*) *transits-turing-machine-loop-cond-false M-def* **by** *simp*
ultimately have *transits M* (*0*, *tps 0*) $(m * t + t1 + 1)$ (*length M1* + *length M2* + *2*, *tps' m*)
using *transits-additive* **by** *fastforce*
then show *?thesis*
using *transforms-def turing-machine-loop-len M-def assms*(*5*) *t-def transits-monotone*
by *auto*
qed

The rule becomes even simpler in the common case in which the Turing machine in the loop head is empty.

lemma *transforms-loop-simple*:
fixes *m t* :: *nat*

```
    and M :: machine
    and tps :: nat ⇒ tape list
  assumes ⋀i. i < m ⟹ transforms M (tps i) t (tps (Suc i))
    and ⋀i. i < m ⟹ cond (read (tps i))
    and ¬ cond (read (tps m))
  assumes ttt ≥ m * (t + 2) + 1
  shows transforms
          (WHILE [] ; cond DO M DONE)
          (tps 0)
          ttt
          (tps m)
  using transforms-loop-for[where ?tps'=tps and ?cond=cond and ?t1.0=0, OF - assms(1) - assms(3)]
    transforms-Nil assms(2,4)
  by simp
```

2.3.5 A proof method

Statements about the behavior and running time of Turing machines, expressed via the predicate *transforms*, are the most common ones we are going to prove. The following proof method applies introduction rules for this predicate. These rules are either the ones we proved for the control structures (sequence, branching, loop) or the ones describing the semantics of concrete Turing machines. The rules of the second kind are collected in the attribute *transforms-intros*.

Applying such a rule usually leaves three kinds of goals: some simple ones requiring only instantiation of schematic variables; one for the equality of two tape lists; and one for the time bound. For the last two goals the proof method offers parameters *tps* and *time*, respectively.

I have to admit that most of the details of the proof method were determined by trial and error.

```
named-theorems transforms-intros
declare transforms-Nil [transforms-intros]

method tform uses tps time declares transforms-intros =
  (
   ((rule transforms-tm-sequentialI)+
    | rule transforms-branchI
    | rule transforms-loop-simple
    | rule transforms-loop-for
    | rule transforms-intros)
   ; (rule transforms-intros)?
   ; (simp only:; fail)?
   ; ((use tps in simp; fail) | (use time in simp; fail))?
   ; (match conclusion in left = right for left right :: tape list
        ⇒ ‹(fastforce simp add: tps list-update-swap; fail)›)?
   ; (match conclusion in left = right for left right :: nat
        ⇒ ‹(use time in simp; fail)?›)?
  )
```

These lemmas are sometimes helpful for proving the equality of tape lists:

```
lemma list-update-swap-less: i' < i ⟹ ys[i := x, i' := x'] = ys[i' := x', i := x]
  by (simp add: list-update-swap)

lemma nth-list-update-neq': j ≠ i ⟹ xs[i := x] ! j = xs ! j
  by simp

end
```

2.4 Elementary Turing machines

```
theory Elementary
  imports Combinations
begin
```

In this section we devise some simple yet useful Turing machines. We have already fully analyzed the empty TM, where start and halting state coincide, in the lemmas *computes-Nil-empty*, *Nil-tm*, and *transforms-Nil*. The next more complex TMs are those with exactly one command. They represent TMs with two states: the halting state and the start state where the action happens. This action might last for one step only, or the TM may stay in this state for longer; for example, it can move a tape head rightward to the next blank symbol. We will also start using the ;; operator to combine some of the one-command TMs.

Most Turing machines we are going to construct throughout this section and indeed the entire article are really families of Turing machines that usually are parameterized by tape indices.

type-synonym $tapeidx = nat$

Throughout this article, names of commands are prefixed with *cmd-* and names of Turing machines with *tm-*. Usually for a TM named *tm-foo* there is a lemma *tm-foo-tm* stating that it really is a Turing machine and a lemma *transforms-tm-fooI* describing its semantics and running time. The lemma usually receives a *transforms-intros* attribute for use with our proof method.

If *tm-foo* comprises more than two TMs we will typically analyze the semantics and running time in a locale named *turing-machine-foo*. The first example of this is *tm-equals* in Section 2.4.10.

When it comes to running times, we will have almost no scruples simplifying upper bounds to have the form $a + b \cdot n^c$ for some constants a, b, c, even if this means, for example, bounding $n \log n$ by n^2.

2.4.1 Clean tapes

Most of our Turing machines will not change the start symbol in the first cell of a tape nor will they write the start symbol anywhere else. The only exceptions are machines that simulate arbitrary other machines. We call tapes that have the start symbol only in the first cell *clean tapes*.

definition $clean\text{-}tape :: tape \Rightarrow bool$ **where**
  $clean\text{-}tape\ tp \equiv \forall i.\ fst\ tp\ i = \triangleright \longleftrightarrow i = 0$

lemma $clean\text{-}tapeI$:
  assumes $\bigwedge i.\ fst\ tp\ i = \triangleright \longleftrightarrow i = 0$
  shows $clean\text{-}tape\ tp$
  unfolding $clean\text{-}tape\text{-}def$ **using** $assms$ **by** $simp$

lemma $clean\text{-}tapeI'$:
  assumes $fst\ tp\ 0 = \triangleright$ **and** $\bigwedge i.\ i > 0 \Longrightarrow fst\ tp\ i \neq \triangleright$
  shows $clean\text{-}tape\ tp$
  unfolding $clean\text{-}tape\text{-}def$ **using** $assms$ **by** $auto$

A clean configuration is one with only clean tapes.

definition $clean\text{-}config :: config \Rightarrow bool$ **where**
  $clean\text{-}config\ cfg \equiv (\forall j{<}||cfg||.\ \forall i.\ (cfg <:> j)\ i = \triangleright \longleftrightarrow i = 0)$

lemma $clean\text{-}config\text{-}tapes$: $clean\text{-}config\ cfg = (\forall tp{\in}set\ (snd\ cfg).\ clean\text{-}tape\ tp)$
  using $clean\text{-}config\text{-}def\ clean\text{-}tape\text{-}def$ **by** $(metis\ in\text{-}set\text{-}conv\text{-}nth)$

lemma $clean\text{-}configI$:
  assumes $\bigwedge j\ i.\ j < length\ tps \Longrightarrow fst\ (tps\ !\ j)\ i = \triangleright \longleftrightarrow i = 0$
  shows $clean\text{-}config\ (q,\ tps)$
  unfolding $clean\text{-}config\text{-}def$ **using** $assms$ **by** $simp$

lemma $clean\text{-}configI'$:
  assumes $\bigwedge tp\ i.\ tp \in set\ tps \Longrightarrow fst\ tp\ i = \triangleright \longleftrightarrow i = 0$
  shows $clean\text{-}config\ (q,\ tps)$
  using $clean\text{-}configI\ assms$ **by** $simp$

lemma $clean\text{-}configI''$:
  assumes $\bigwedge tp.\ tp \in set\ tps \Longrightarrow (fst\ tp\ 0 = \triangleright \wedge (\forall i{>}0.\ fst\ tp\ i \neq \triangleright))$
  shows $clean\text{-}config\ (q,\ tps)$
  using $clean\text{-}configI'\ assms$ **by** $blast$

lemma *clean-configE*:
  assumes *clean-config* (q, tps)
  shows $\bigwedge j\ i.\ j < \mathit{length}\ tps \Longrightarrow \mathit{fst}\ (tps\ !\ j)\ i = \triangleright \longleftrightarrow i = 0$
  using *clean-config-def assms* **by** *simp*

lemma *clean-configE′*:
  assumes *clean-config* (q, tps)
  shows $\bigwedge tp\ i.\ tp \in \mathit{set}\ tps \Longrightarrow \mathit{fst}\ tp\ i = \triangleright \longleftrightarrow i = 0$
  using *clean-configE assms* **by** (*metis in-set-conv-nth*)

lemma *clean-contents-proper* [*simp*]: *proper-symbols zs* $\Longrightarrow$ *clean-tape* $(\lfloor zs \rfloor, q)$
  using *clean-tape-def contents-def proper-symbols-ne1* **by** *simp*

lemma *contents-clean-tape′*: *proper-symbols zs* $\Longrightarrow$ *fst tp* $= \lfloor zs \rfloor \Longrightarrow$ *clean-tape tp*
  using *contents-def clean-tape-def* **by** (*simp add*: *nat-neq-iff*)

Some more lemmas about *contents*:

lemma *contents-append-blanks*: $\lfloor ys\ @\ \mathit{replicate}\ m\ \Box \rfloor = \lfloor ys \rfloor$
proof
  fix *i*
  consider
      $i = 0$
    $|\ 0 < i \wedge i \leq \mathit{length}\ ys$
    $|\ \mathit{length}\ ys < i \wedge i \leq \mathit{length}\ ys + m$
    $|\ \mathit{length}\ ys + m < i$
    by *linarith*
  then show $\lfloor ys\ @\ \mathit{replicate}\ m\ \Box \rfloor\ i = \lfloor ys \rfloor\ i$
  proof (*cases*)
    case *1*
    then show *?thesis*
      by *simp*
  next
    case *2*
    then show *?thesis*
      using *contents-inbounds*
      by (*metis* (*no-types*, *opaque-lifting*) *Suc-diff-1 Suc-le-eq le-add1 le-trans length-append nth-append*)
  next
    case *3*
    then show *?thesis*
      by (*smt* (*verit*) *Suc-diff-Suc add-diff-inverse-nat contents-def diff-Suc-1 diff-commute leD less-one less-or-eq-imp-le nat-add-left-cancel-le not-less-eq nth-append nth-replicate*)
  next
    case *4*
    then show *?thesis*
      by *simp*
  qed
qed

lemma *contents-append-update*:
  assumes $\mathit{length}\ ys = m$
  shows $\lfloor ys\ @\ [v]\ @\ zs \rfloor(\mathit{Suc}\ m := w) = \lfloor ys\ @\ [w]\ @\ zs \rfloor$
proof
  fix *i*
  consider
      $i = 0$
    $|\ 0 < i \wedge i < \mathit{Suc}\ m$
    $|\ i = \mathit{Suc}\ m$
    $|\ i > \mathit{Suc}\ m \wedge i \leq \mathit{Suc}\ m + \mathit{length}\ zs$
    $|\ i > \mathit{Suc}\ m + \mathit{length}\ zs$
    by *linarith*
  then show $(\lfloor ys\ @\ [v]\ @\ zs \rfloor(\mathit{Suc}\ m := w))\ i = \lfloor ys\ @\ [w]\ @\ zs \rfloor\ i$
    (**is** $?l = ?r$)

proof (*cases*)
case *1*
then show *?thesis*
by *simp*
next
case *2*
then have *?l* = (*ys* @ [*v*] @ *zs*) ! (*i* − *1*)
using *assms contents-inbounds* **by** *simp*
then have ∗: *?l* = *ys* ! (*i* − *1*)
using *2 assms* **by** (*metis Suc-diff-1 Suc-le-lessD less-Suc-eq-le nth-append*)
have *?r* = (*ys* @ [*w*] @ *zs*) ! (*i* − *1*)
using *2 assms contents-inbounds* **by** *simp*
then have *?r* = *ys* ! (*i* − *1*)
using *2 assms* **by** (*metis Suc-diff-1 Suc-le-lessD less-Suc-eq-le nth-append*)
then show *?thesis*
using ∗ **by** *simp*
next
case *3*
then show *?thesis*
using *assms* **by** *auto*
next
case *4*
then have *?l* = (*ys* @ [*v*] @ *zs*) ! (*i* − *1*)
using *assms contents-inbounds* **by** *simp*
then have *?l* = ((*ys* @ [*v*]) @ *zs*) ! (*i* − *1*)
by *simp*
then have ∗: *?l* = *zs* ! (*i* − *1* − *Suc m*)
using *4 assms* **by** (*metis diff-Suc-1 length-append-singleton less-imp-Suc-add not-add-less1 nth-append*)
then have *?r* = (*ys* @ [*w*] @ *zs*) ! (*i* − *1*)
using *4 assms contents-inbounds* **by** *simp*
then have *?r* = ((*ys* @ [*w*]) @ *zs*) ! (*i* − *1*)
by *simp*
then have *?r* = *zs* ! (*i* − *1* − *Suc m*)
using *4 assms* **by** (*metis diff-Suc-1 length-append-singleton less-imp-Suc-add not-add-less1 nth-append*)
then show *?thesis*
using ∗ **by** *simp*
next
case *5*
then show *?thesis*
using *assms* **by** *simp*
qed
qed

lemma *contents-snoc*: ⌊*ys*⌋(*Suc* (*length ys*) := *w*) = ⌊*ys* @ [*w*]⌋
proof
fix *i*
consider *i* = *0* | *0* < *i* ∧ *i* ≤ *length ys* | *i* = *Suc* (*length ys*) | *i* > *Suc* (*length ys*)
by *linarith*
then show (⌊*ys*⌋(*Suc* (*length ys*) := *w*)) *i* = ⌊*ys* @ [*w*]⌋ *i*
proof (*cases*)
case *1*
then show *?thesis*
by *simp*
next
case *2*
then show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *contents-def diff-Suc-1 ex-least-nat-less fun-upd-apply leD le-Suc-eq length-append-singleton less-imp-Suc-add nth-append*)
next
case *3*
then show *?thesis*
by *simp*
next

case *4*
then show *?thesis*
by *simp*
qed
qed

definition *config-update-pos* :: *config* ⇒ *nat* ⇒ *nat* ⇒ *config* **where**
config-update-pos cfg j p ≡ (*fst cfg*, (*snd cfg*)[*j*:=(*cfg* <:> *j*, *p*)])

lemma *config-update-pos-0*: *config-update-pos cfg j* (*cfg* <#> *j*) = *cfg*
using *config-update-pos-def* **by** *simp*

definition *config-update-fwd* :: *config* ⇒ *nat* ⇒ *nat* ⇒ *config* **where**
config-update-fwd cfg j d ≡ (*fst cfg*, (*snd cfg*)[*j*:=(*cfg* <:> *j*, *cfg* <#> *j* + *d*)])

lemma *config-update-fwd-0*: *config-update-fwd cfg j 0* = *cfg*
using *config-update-fwd-def* **by** *simp*

lemma *config-update-fwd-additive*:
config-update-fwd (*config-update-fwd cfg j d1*) *j d2* = (*config-update-fwd cfg j* (*d1* + *d2*))
using *config-update-fwd-def*
by (*smt* (*verit*) *add.commute add.left-commute fst-conv le-less-linear list-update-beyond list-update-overwrite nth-list-update-eq sndI*)

2.4.2 Moving tape heads

Among the most simple things a Turing machine can do is moving one of its tape heads.

Moving left

The next command makes a TM move its head on tape *j* one cell to the left unless, of course, it is in the leftmost cell already.

definition *cmd-left* :: *tapeidx* ⇒ *command* **where**
cmd-left j ≡ λ*rs*. (*1*, *map* (λ*i*. (*rs* ! *i*, *if i* = *j then Left else Stay*)) [*0*..<*length rs*])

lemma *turing-command-left*: *turing-command k 1 G* (*cmd-left j*)
by (*auto simp add*: *cmd-left-def*)

lemma *cmd-left′*: [∗] (*cmd-left j rs*) = *1*
using *cmd-left-def* **by** *simp*

lemma *cmd-left″*: *j* < *length rs* ⟹ (*cmd-left j rs*) [!] *j* = (*rs* ! *j*, *Left*)
using *cmd-left-def* **by** *simp*

lemma *cmd-left‴*: *i* < *length rs* ⟹ *i* ≠ *j* ⟹ (*cmd-left j rs*) [!] *i* = (*rs* ! *i*, *Stay*)
using *cmd-left-def* **by** *simp*

lemma *tape-list-eq*:
assumes *length tps′* = *length tps*
and ⋀*i*. *i* < *length tps* ⟹ *i* ≠ *j* ⟹ *tps′* ! *i* = *tps* ! *i*
and *tps′* ! *j* = *x*
shows *tps′* = *tps*[*j* := *x*]
using *assms* **by** (*smt* (*verit*) *length-list-update list-update-beyond not-le nth-equalityI nth-list-update*)

lemma *sem-cmd-left*:
assumes *j* < *length tps*
shows *sem* (*cmd-left j*) (*0*, *tps*) = (*1*, *tps*[*j*:=(*fst* (*tps* ! *j*), *snd* (*tps* ! *j*) − *1*)])
proof
show *fst* (*sem* (*cmd-left j*) (*0*, *tps*)) = *fst* (*1*, *tps*[*j* := (*fst* (*tps* ! *j*), *snd* (*tps* ! *j*) − *1*)])
using *cmd-left′ sem-fst* **by** *simp*
have *snd* (*sem* (*cmd-left j*) (*0*, *tps*)) = *tps*[*j* := (*fst* (*tps* ! *j*), *snd* (*tps* ! *j*) − *1*)]
proof (*rule tape-list-eq*)

show ||*sem* (*cmd-left j*) (*0*, *tps*)|| = *length tps*
using *turing-command-left sem-num-tapes2'* **by** (*metis snd-eqD*)
show *sem* (*cmd-left j*) (*0*, *tps*) <!> *i* = *tps* ! *i* **if** *i* < *length tps* **and** $i \neq j$ **for** *i*
proof −
let *?rs* = *read tps*
have *length ?rs* = *length tps*
using *read-length* **by** *simp*
moreover have *sem* (*cmd-left j*) (*0*, *tps*) <!> *i* = *act* (*cmd-left j ?rs* [!] *i*) (*tps* ! *i*)
by (*simp add*: *cmd-left-def sem-snd that*(*1*))
ultimately show *?thesis*
using *that act-Stay cmd-left'''* **by** *simp*
qed
show *sem* (*cmd-left j*) (*0*, *tps*) <!> *j* = (*fst* (*tps* ! *j*), *snd* (*tps* ! *j*) − *1*)
using *assms act-Left cmd-left-def read-length sem-snd* **by** *simp*
qed
then show *snd* (*sem* (*cmd-left j*) (*0*, *tps*)) = *snd* (*1*, *tps*[*j* := (*fst* (*tps* ! *j*), *snd* (*tps* ! *j*) − *1*)])
by *simp*
qed

definition *tm-left* :: *tapeidx* ⇒ *machine* **where**
tm-left j ≡ [*cmd-left j*]

lemma *tm-left-tm*: $k \geq 2 \Longrightarrow G \geq 4 \Longrightarrow$ *turing-machine k G* (*tm-left j*)
unfolding *tm-left-def* **using** *turing-command-left* **by** *auto*

lemma *exe-tm-left*:
assumes *j* < *length tps*
shows *exe* (*tm-left j*) (*0*, *tps*) = (*1*, *tps*[*j* := *tps* ! *j* |−| *1*])
unfolding *tm-left-def* **using** *sem-cmd-left assms* **by** (*simp add*: *exe-lt-length*)

lemma *execute-tm-left*:
assumes *j* < *length tps*
shows *execute* (*tm-left j*) (*0*, *tps*) (*Suc 0*) = (*1*, *tps*[*j* := *tps* ! *j* |−| *1*])
using *assms exe-tm-left* **by** *simp*

lemma *transits-tm-left*:
assumes *j* < *length tps*
shows *transits* (*tm-left j*) (*0*, *tps*) (*Suc 0*) (*1*, *tps*[*j* := *tps* ! *j* |−| *1*])
using *execute-tm-left assms transitsI* **by** *blast*

lemma *transforms-tm-left*:
assumes *j* < *length tps*
shows *transforms* (*tm-left j*) *tps* (*Suc 0*) (*tps*[*j* := *tps* ! *j* |−| *1*])
using *transits-tm-left assms* **by** (*simp add*: *tm-left-def transforms-def*)

lemma *transforms-tm-leftI* [*transforms-intros*]:
assumes *j* < *length tps*
and *t* = *1*
and *tps'* = *tps*[*j* := *tps* ! *j* |−| *1*]
shows *transforms* (*tm-left j*) *tps t tps'*
using *transits-tm-left assms* **by** (*simp add*: *tm-left-def transforms-def*)

Moving right

The next command makes the head on tape *j* move one cell to the right.

definition *cmd-right* :: *tapeidx* ⇒ *command* **where**
cmd-right j ≡ λ*rs*. (*1*, *map* (λ*i*. (*rs* ! *i*, *if i* = *j then Right else Stay*)) [*0*..<*length rs*])

lemma *turing-command-right*: *turing-command k 1 G* (*cmd-right j*)
by (*auto simp add*: *cmd-right-def*)

lemma *cmd-right'*: [∗] (*cmd-right j rs*) = *1*
using *cmd-right-def* **by** *simp*

```
lemma cmd-right'': j < length rs ⟹ (cmd-right j rs) [!] j = (rs ! j, Right)
  using cmd-right-def by simp

lemma cmd-right''': i < length rs ⟹ i ≠ j ⟹ (cmd-right j rs) [!] i = (rs ! i, Stay)
  using cmd-right-def by simp

lemma sem-cmd-right:
  assumes j < length tps
  shows sem (cmd-right j) (0, tps) = (1, tps[j:=(fst (tps ! j), snd (tps ! j) + 1)])
proof
  show fst (sem (cmd-right j)  (0, tps)) = fst (1, tps[j := (fst (tps ! j), snd (tps ! j) + 1)])
    using cmd-right' sem-fst by simp
  have snd (sem (cmd-right j) (0, tps)) = tps[j := (fst (tps ! j), snd (tps ! j) + 1)]
  proof (rule tape-list-eq)
    show ||sem (cmd-right j) (0, tps)|| = length tps
      using sem-num-tapes3 turing-command-right by (metis snd-conv)
    show sem (cmd-right j) (0, tps) <!> i = tps ! i if i < length tps and i ≠ j for i
    proof −
      let ?rs = read tps
      have length ?rs = length tps
        using read-length by simp
      moreover have sem (cmd-right j) (0, tps) <!> i = act (cmd-right j ?rs [!] i) (tps ! i)
        by (simp add: cmd-right-def sem-snd that(1))
      ultimately show ?thesis
        using that act-Stay cmd-right''' by simp
    qed
    show sem (cmd-right j) (0, tps) <!> j = (fst (tps ! j), snd (tps ! j) + 1)
      using assms act-Right cmd-right-def read-length sem-snd by simp
  qed
  then show snd (sem (cmd-right j) (0, tps)) = snd (1, tps[j := (fst (tps ! j), snd (tps ! j) + 1)])
    by simp
qed

definition tm-right :: tapeidx ⇒ machine where
  tm-right j ≡ [cmd-right j]

lemma tm-right-tm: k ≥ 2 ⟹ G ≥ 4 ⟹ turing-machine k G (tm-right j)
  unfolding tm-right-def using turing-command-right cmd-right' by auto

lemma exe-tm-right:
  assumes j < length tps
  shows exe (tm-right j) (0, tps) = (1, tps[j:=(fst (tps ! j), snd (tps ! j) + 1)])
  unfolding tm-right-def using sem-cmd-right assms by (simp add: exe-lt-length)

lemma execute-tm-right:
  assumes j < length tps
  shows execute (tm-right j) (0, tps) (Suc 0) = (1, tps[j:=(fst (tps ! j), snd (tps ! j) + 1)])
  using assms exe-tm-right by simp

lemma transits-tm-right:
  assumes j < length tps
  shows transits (tm-right j) (0, tps) (Suc 0) (1, tps[j:=(fst (tps ! j), snd (tps ! j) + 1)])
  using execute-tm-right assms transitsI by blast

lemma transforms-tm-right:
  assumes j < length tps
  shows transforms (tm-right j) tps (Suc 0) (tps[j := tps ! j |+| 1])
  using transits-tm-right assms by (simp add: tm-right-def transforms-def)

lemma transforms-tm-rightI [transforms-intros]:
  assumes j < length tps
    and t = Suc 0
```

```
    and tps' = tps[j := tps ! j |+| 1]
  shows transforms (tm-right j) tps t tps'
  using transits-tm-right assms by (simp add: tm-right-def transforms-def)
```

Moving right on several tapes

The next command makes the heads on all tapes from a set J of tapes move one cell to the right.

```
definition cmd-right-many :: tapeidx set ⇒ command where
  cmd-right-many J ≡ λrs. (1, map (λi. (rs ! i, if i ∈ J then Right else Stay)) [0..<length rs])

lemma turing-command-right-many: turing-command k 1 G (cmd-right-many J)
  by (auto simp add: cmd-right-many-def)

lemma sem-cmd-right-many:
  sem (cmd-right-many J) (0, tps) = (1, map (λj. if j ∈ J then tps ! j |+| 1 else tps ! j) [0..<length tps])
proof
  show fst (sem (cmd-right-many J)  (0, tps)) =
      fst (1, map (λj. if j ∈ J then tps ! j |+| 1 else tps ! j) [0..<length tps])
    using cmd-right-many-def sem-fst by simp
  have snd (sem (cmd-right-many J) (0, tps)) =
    map (λj. if j ∈ J then tps ! j |+| 1 else tps ! j) [0..<length tps]
    (is ?lhs = ?rhs)
  proof (rule nth-equalityI)
    show length ?lhs = length ?rhs
      using turing-command-right-many sem-num-tapes2'
      by (metis (no-types, lifting) diff-zero length-map length-upt snd-conv)
    then have len: length ?lhs = length tps
      by simp
    show ?lhs ! j = ?rhs ! j if j < length ?lhs for j
    proof (cases j ∈ J)
      case True
      let ?rs = read tps
      have length ?rs = length tps
        using read-length by simp
      moreover have sem (cmd-right-many J) (0, tps) <!> j = act (cmd-right-many J ?rs [!] j) (tps ! j)
        using cmd-right-many-def sem-snd that True len by auto
      moreover have ?rhs ! j = tps ! j |+| 1
        using that len True by simp
      ultimately show ?thesis
        using that act-Right cmd-right-many-def True len by simp
    next
      case False
      let ?rs = read tps
      have length ?rs = length tps
        using read-length by simp
      moreover have sem (cmd-right-many J) (0, tps) <!> j = act (cmd-right-many J ?rs [!] j) (tps ! j)
        using cmd-right-many-def sem-snd that False len by auto
      moreover have ?rhs ! j = tps ! j
        using that len False by simp
      ultimately show ?thesis
        using that act-Stay cmd-right-many-def False len by simp
    qed
  qed
  then show snd (sem (cmd-right-many J) (0, tps)) =
      snd (1, map (λj. if j ∈ J then tps ! j |+| 1 else tps ! j) [0..<length tps])
    by simp
qed

definition tm-right-many :: tapeidx set ⇒ machine where
  tm-right-many J ≡ [cmd-right-many J]

lemma tm-right-many-tm: k ≥ 2 ⟹ G ≥ 4 ⟹ turing-machine k G (tm-right-many J)
  unfolding tm-right-many-def using turing-command-right-many by auto
```

lemma *transforms-tm-right-manyI* [*transforms-intros*]:
  assumes $t = Suc\ 0$
    and $tps' = map\ (\lambda j.\ if\ j \in J\ then\ tps\ !\ j\ |+|\ 1\ else\ tps\ !\ j)\ [0..<length\ tps]$
  shows *transforms* (*tm-right-many J*) *tps t tps'*
proof −
  have $exe\ (tm\text{-}right\text{-}many\ J)\ (0,\ tps) = (1,\ map\ (\lambda j.\ if\ j \in J\ then\ tps\ !\ j\ |+|\ 1\ else\ tps\ !\ j)\ [0..<length\ tps])$
    unfolding *tm-right-many-def* **using** *sem-cmd-right-many* **by** (*simp add*: *exe-lt-length*)
  then have $execute\ (tm\text{-}right\text{-}many\ J)\ (0,\ tps)\ (Suc\ 0) = (1,\ map\ (\lambda j.\ if\ j \in J\ then\ tps\ !\ j\ |+|\ 1\ else\ tps\ !\ j)\ [0..<length\ tps])$
    by *simp*
  then have $transits\ (tm\text{-}right\text{-}many\ J)\ (0,\ tps)\ (Suc\ 0)\ (1,\ map\ (\lambda j.\ if\ j \in J\ then\ tps\ !\ j\ |+|\ 1\ else\ tps\ !\ j)\ [0..<length\ tps])$
    using *transitsI* **by** *blast*
  then have $transforms\ (tm\text{-}right\text{-}many\ J)\ tps\ (Suc\ 0)\ (map\ (\lambda j.\ if\ j \in J\ then\ tps\ !\ j\ |+|\ 1\ else\ tps\ !\ j)\ [0..<length\ tps])$
    by (*simp add*: *tm-right-many-def transforms-def*)
  then show *?thesis*
    using *assms* **by** (*simp add*: *tm-right-many-def transforms-def*)
qed

2.4.3 Copying and translating tape contents

The Turing machines in this section scan a tape j_1 and copy the symbols to another tape j_2. Scanning can be performed in either direction, and "copying" may include mapping the symbols.

Copying and translating from one tape to another

The next predicate is true iff. on the given tape the next symbol from the set H of symbols is exactly n cells to the right from the current head position. Thus, a command that moves the tape head right until it finds a symbol from H takes n steps and moves the head n cells right.

definition *rneigh* :: *tape* ⇒ *symbol set* ⇒ *nat* ⇒ *bool* **where**
  $rneigh\ tp\ H\ n \equiv fst\ tp\ (snd\ tp + n) \in H \wedge (\forall n' < n.\ fst\ tp\ (snd\ tp + n') \notin H)$

lemma *rneighI*:
  assumes $fst\ tp\ (snd\ tp + n) \in H$ **and** $\bigwedge n'.\ n' < n \Longrightarrow fst\ tp\ (snd\ tp + n') \notin H$
  shows *rneigh tp H n*
  using *assms rneigh-def* **by** *simp*

The analogous predicate for moving to the left:

definition *lneigh* :: *tape* ⇒ *symbol set* ⇒ *nat* ⇒ *bool* **where**
  $lneigh\ tp\ H\ n \equiv fst\ tp\ (snd\ tp - n) \in H \wedge (\forall n' < n.\ fst\ tp\ (snd\ tp - n') \notin H)$

lemma *lneighI*:
  assumes $fst\ tp\ (snd\ tp - n) \in H$ **and** $\bigwedge n'.\ n' < n \Longrightarrow fst\ tp\ (snd\ tp - n') \notin H$
  shows *lneigh tp H n*
  using *assms lneigh-def* **by** *simp*

The next command scans tape j_1 rightward until it reaches a symbol from the set H. While doing so it copies the symbols, after applying a mapping f, to tape j_2.

definition *cmd-trans-until* :: *tapeidx* ⇒ *tapeidx* ⇒ *symbol set* ⇒ (*symbol* ⇒ *symbol*) ⇒ *command* **where**
  $cmd\text{-}trans\text{-}until\ j1\ j2\ H\ f \equiv \lambda rs.$
    $if\ rs\ !\ j1 \in H$
    $then\ (1,\ map\ (\lambda r.\ (r,\ Stay))\ rs)$
    $else\ (0,\ map\ (\lambda i.\ (if\ i = j2\ then\ f\ (rs\ !\ j1)\ else\ rs\ !\ i,\ if\ i = j1 \vee i = j2\ then\ Right\ else\ Stay))\ [0..<length\ rs])$

The analogous command for moving to the left:

definition *cmd-ltrans-until* :: *tapeidx* ⇒ *tapeidx* ⇒ *symbol set* ⇒ (*symbol* ⇒ *symbol*) ⇒ *command* **where**
  $cmd\text{-}ltrans\text{-}until\ j1\ j2\ H\ f \equiv \lambda rs.$
    $if\ rs\ !\ j1 \in H$
    $then\ (1,\ map\ (\lambda r.\ (r,\ Stay))\ rs)$

else $(0, map\ (\lambda i.\ (if\ i = j2\ then\ f\ (rs\ !\ j1)\ else\ rs\ !\ i,\ if\ i = j1 \lor i = j2\ then\ Left\ else\ Stay))\ [0..<length\ rs])$

lemma *proper-cmd-trans-until*: *proper-command k* (*cmd-trans-until j1 j2 H f*)
using *cmd-trans-until-def* **by** *simp*

lemma *proper-cmd-ltrans-until*: *proper-command k* (*cmd-ltrans-until j1 j2 H f*)
using *cmd-ltrans-until-def* **by** *simp*

lemma *sem-cmd-trans-until-1*:
assumes $j1 < k$ **and** *length tps* $= k$ **and** $(0, tps) <.> j1 \in H$
shows *sem* (*cmd-trans-until j1 j2 H f*) $(0, tps) = (1, tps)$
using *cmd-trans-until-def tapes-at-read read-length assms act-Stay*
by (*intro semI*[*OF proper-cmd-trans-until*]) *auto*

lemma *sem-cmd-ltrans-until-1*:
assumes $j1 < k$ **and** *length tps* $= k$ **and** $(0, tps) <.> j1 \in H$
shows *sem* (*cmd-ltrans-until j1 j2 H f*) $(0, tps) = (1, tps)$
using *cmd-ltrans-until-def tapes-at-read read-length assms act-Stay*
by (*intro semI*[*OF proper-cmd-ltrans-until*]) *auto*

lemma *sem-cmd-trans-until-2*:
assumes $j1 < k$ **and** *length tps* $= k$ **and** $(0, tps) <.> j1 \notin H$
shows *sem* (*cmd-trans-until j1 j2 H f*) $(0, tps) =$
$(0, tps[j1 := tps\ !\ j1\ |+|\ 1, j2 := tps\ !\ j2\ |{:=}|\ (f\ (tps\ {:.:}\ j1))\ |+|\ 1])$
using *cmd-trans-until-def tapes-at-read read-length assms act-Stay act-Right*
by (*intro semI*[*OF proper-cmd-trans-until*]) *auto*

lemma *sem-cmd-ltrans-until-2*:
assumes $j1 < k$ **and** *length tps* $= k$ **and** $(0, tps) <.> j1 \notin H$
shows *sem* (*cmd-ltrans-until j1 j2 H f*) $(0, tps) =$
$(0, tps[j1 := tps\ !\ j1\ |-|\ 1, j2 := tps\ !\ j2\ |{:=}|\ (f\ (tps\ {:.:}\ j1))\ |-|\ 1])$
using *cmd-ltrans-until-def tapes-at-read read-length assms act-Stay act-Left*
by (*intro semI*[*OF proper-cmd-ltrans-until*]) *auto*

definition *tm-trans-until* :: *tapeidx* $\Rightarrow$ *tapeidx* $\Rightarrow$ *symbol set* $\Rightarrow$ (*symbol* $\Rightarrow$ *symbol*) $\Rightarrow$ *machine* **where**
tm-trans-until j1 j2 H f $\equiv$ [*cmd-trans-until j1 j2 H f*]

definition *tm-ltrans-until* :: *tapeidx* $\Rightarrow$ *tapeidx* $\Rightarrow$ *symbol set* $\Rightarrow$ (*symbol* $\Rightarrow$ *symbol*) $\Rightarrow$ *machine* **where**
tm-ltrans-until j1 j2 H f $\equiv$ [*cmd-ltrans-until j1 j2 H f*]

lemma *tm-trans-until-tm*:
assumes $0 < j2$ **and** $j1 < k$ **and** $j2 < k$ **and** $\forall h<G.\ f\ h < G$ **and** $k \geq 2$ **and** $G \geq 4$
shows *turing-machine k G* (*tm-trans-until j1 j2 H f*)
unfolding *tm-trans-until-def cmd-trans-until-def* **using** *assms turing-machine-def* **by** *auto*

lemma *tm-ltrans-until-tm*:
assumes $0 < j2$ **and** $j1 < k$ **and** $j2 < k$ **and** $\forall h<G.\ f\ h < G$ **and** $k \geq 2$ **and** $G \geq 4$
shows *turing-machine k G* (*tm-ltrans-until j1 j2 H f*)
unfolding *tm-ltrans-until-def cmd-ltrans-until-def* **using** *assms turing-machine-def* **by** *auto*

lemma *exe-tm-trans-until-1*:
assumes $j1 < k$ **and** *length tps* $= k$ **and** $(0, tps) <.> j1 \in H$
shows *exe* (*tm-trans-until j1 j2 H f*) $(0, tps) = (1, tps)$
unfolding *tm-trans-until-def* **using** *sem-cmd-trans-until-1 assms exe-lt-length* **by** *simp*

lemma *exe-tm-ltrans-until-1*:
assumes $j1 < k$ **and** *length tps* $= k$ **and** $(0, tps) <.> j1 \in H$
shows *exe* (*tm-ltrans-until j1 j2 H f*) $(0, tps) = (1, tps)$
unfolding *tm-ltrans-until-def* **using** *sem-cmd-ltrans-until-1 assms exe-lt-length* **by** *simp*

lemma *exe-tm-trans-until-2*:
assumes $j1 < k$ **and** *length tps* $= k$ **and** $(0, tps) <.> j1 \notin H$
shows *exe* (*tm-trans-until j1 j2 H f*) $(0, tps) =$

$(0, tps[j1 := tps ! j1 |+| 1, j2 := tps ! j2 |:=| (f (tps :.: j1)) |+| 1])$
unfolding *tm-trans-until-def* **using** *sem-cmd-trans-until-2 assms exe-lt-length* **by** *simp*

lemma *exe-tm-ltrans-until-2*:
assumes $j1 < k$ **and** *length tps* $= k$ **and** $(0, tps) <.> j1 \notin H$
shows *exe* (*tm-ltrans-until j1 j2 H f*) $(0, tps) =$
$(0, tps[j1 := tps ! j1 |-| 1, j2 := tps ! j2 |:=| (f (tps :.: j1)) |-| 1])$
unfolding *tm-ltrans-until-def* **using** *sem-cmd-ltrans-until-2 assms exe-lt-length* **by** *simp*

Let tp_1 and tp_2 be two tapes with head positions i_1 and i_2, respectively. The next function describes the result of overwriting the symbols at positions $i_2, \ldots, i_2 + n - 1$ on tape tp_2 by the symbols at positions $i_1, \ldots, i_1 + n - 1$ on tape tp_1 after applying a symbol map f.

definition *transplant* :: *tape* $\Rightarrow$ *tape* $\Rightarrow$ (*symbol* $\Rightarrow$ *symbol*)$\Rightarrow$ *nat* $\Rightarrow$ *tape* **where**
transplant tp1 tp2 f n $\equiv$
($\lambda i.$ *if snd tp2* $\leq i \wedge i <$ *snd tp2* $+ n$ *then f* (*fst tp1* (*snd tp1* $+ i -$ *snd tp2*)) *else fst tp2 i*,
snd tp2 $+ n$)

The analogous function for moving to the left while copying:

definition *ltransplant* :: *tape* $\Rightarrow$ *tape* $\Rightarrow$ (*symbol* $\Rightarrow$ *symbol*)$\Rightarrow$ *nat* $\Rightarrow$ *tape* **where**
ltransplant tp1 tp2 f n $\equiv$
($\lambda i.$ *if snd tp2* $- n < i \wedge i \leq$ *snd tp2 then f* (*fst tp1* (*snd tp1* $+ i -$ *snd tp2*)) *else fst tp2 i*,
snd tp2 $- n$)

lemma *transplant-0*: *transplant tp1 tp2 f 0* $=$ *tp2*
unfolding *transplant-def* **by** *standard auto*

lemma *ltransplant-0*: *ltransplant tp1 tp2 f 0* $=$ *tp2*
unfolding *ltransplant-def* **by** *standard auto*

lemma *transplant-upd*: *transplant tp1 tp2 f n* $|{:=}|$ (*f* ($|.|$ (*tp1* $|+|$ *n*))) $|+|$ *1* $=$ *transplant tp1 tp2 f* (*Suc n*)
unfolding *transplant-def* **by** *auto*

lemma *ltransplant-upd*:
assumes $n <$ *snd tp2*
shows *ltransplant tp1 tp2 f n* $|{:=}|$ (*f* ($|.|$ (*tp1* $|-|$ *n*))) $|-|$ *1* $=$ *ltransplant tp1 tp2 f* (*Suc n*)
unfolding *ltransplant-def* **using** *assms* **by** *fastforce*

lemma *tapes-ltransplant-upd*:
assumes $t <$ *tps* :#: *j2* **and** $t <$ *tps* :#: *j1* **and** $j1 < k$ **and** $j2 < k$ **and** *length tps* $= k$
and $tps' = tps[j1 := tps ! j1 |-| t, j2 :=$ *ltransplant* (*tps* ! *j1*) (*tps* ! *j2*) *f t*]
shows $tps'[j1 := tps' ! j1 |-| 1, j2 := tps' ! j2 |:=| (f (tps' :.: j1)) |-| 1] =$
$tps[j1 := tps ! j1 |-|$ *Suc t*, $j2 :=$ *ltransplant* (*tps* ! *j1*) (*tps* ! *j2*) *f* (*Suc t*)]
(**is** *?lhs* $=$ *?rhs*)
proof (*rule nth-equalityI*)
show *1*: *length ?lhs* $=$ *length ?rhs*
using *assms* **by** *simp*
have *len*: *length ?lhs* $= k$
using *assms* **by** *simp*
show *?lhs* ! $j =$ *?rhs* ! j **if** $j <$ *length ?lhs* **for** j
proof $-$
have $j < k$
using *len that* **by** *simp*
show *?thesis*
proof (*cases* $j \neq j1 \wedge j \neq j2$)
case *True*
then show *?thesis*
using *assms* **by** *simp*
next
case *j1j2*: *False*
show *?thesis*
proof (*cases* $j1 = j2$)
case *True*
then have *j*: $j = j1$ $j = j2$

using *j1j2* **by** *simp-all*
have tps' ! *j1* = *ltransplant* (*tps* ! *j1*) (*tps* ! *j2*) *f t*
using *j assms that* **by** *simp*
then have ∗: *snd* (tps' ! *j1*) = *snd* (*tps* ! *j1*) − *t*
using *j ltransplant-def* **by** *simp*
then have *fst* (tps' ! *j1*) =
(λi. *if snd* (*tps* ! *j2*) − *t* < *i* ∧ *i* ≤ *snd* (*tps* ! *j2*) *then f* (*fst* (*tps* ! *j1*) (*snd* (*tps* ! *j1*) + *i* − *snd* (*tps* ! *j2*))) *else fst* (*tps* ! *j2*) *i*)
using *j ltransplant-def assms* **by** *auto*
then have *fst* (tps' ! *j1*) =
(λi. *if snd* (*tps* ! *j1*) − *t* < *i* ∧ *i* ≤ *snd* (*tps* ! *j1*) *then f* (*fst* (*tps* ! *j1*) (*snd* (*tps* ! *j1*) + *i* − *snd* (*tps* ! *j1*))) *else fst* (*tps* ! *j1*) *i*)
using *j* **by** *auto*
then have *fst* (tps' ! *j1*) (*snd* (*tps* ! *j1*) − *t*) = *fst* (*tps* ! *j1*) (*snd* (*tps* ! *j1*) − *t*)
by *simp*
then have tps' :.: *j1* = *fst* (*tps* ! *j1*) (*snd* (*tps* ! *j1*) − *t*)
using ∗ **by** *simp*
then have *?lhs* ! *j* = (*ltransplant* (*tps* ! *j1*) (*tps* ! *j2*) *f t*) |:=| (*f* (|.| (*tps* ! *j1* |−| *t*))) |−| *1*
using *assms*(*6*) *j that* **by** *simp*
then have *?lhs* ! *j* = (*ltransplant* (*tps* ! *j1*) (*tps* ! *j2*) *f* (*Suc t*))
using *ltransplant-upd assms*(*1*) **by** *simp*
moreover have *?rhs* ! *j* = *ltransplant* (*tps* ! *j1*) (*tps* ! *j2*) *f* (*Suc t*)
using *assms*(*6*) *that j* **by** *simp*
ultimately show *?thesis*
by *simp*
next
case *j1-neq-j2*: *False*
then show *?thesis*
proof (*cases j* = *j1*)
case *True*
then have *?lhs* ! *j* = tps' ! *j1* |−| *1*
using *assms j1-neq-j2* **by** *simp*
then have *?lhs* ! *j* = (*tps* ! *j1* |−| *t*) |−| *1*
using *assms j1-neq-j2* **by** *simp*
moreover have *?rhs* ! *j* = *tps* ! *j1* |−| *Suc t*
using *True assms j1-neq-j2* **by** *simp*
ultimately show *?thesis*
by *simp*
next
case *False*
then have *j*: *j* = *j2*
using *j1j2* **by** *simp*
then have *?lhs* ! *j* = tps' ! *j2* |:=| (*f* (tps' :.: *j1*)) |−| *1*
using *assms* **by** *simp*
then have *?lhs* ! *j* = (*ltransplant* (*tps* ! *j1*) (*tps* ! *j2*) *f t*) |:=| (*f* (tps' :.: *j1*)) |−| *1*
using *assms* **by** *simp*
then have *?lhs* ! *j* = (*ltransplant* (*tps* ! *j1*) (*tps* ! *j2*) *f* (*Suc t*))
using *ltransplant-def assms ltransplant-upd* **by** (*smt* (*verit*) *j1-neq-j2 nth-list-update-eq nth-list-update-neq*)
moreover have *?rhs* ! *j* = *ltransplant* (*tps* ! *j1*) (*tps* ! *j2*) *f* (*Suc t*)
using *assms*(*6*) *that j* **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed
qed
qed
qed

lemma *execute-tm-trans-until-less*:
assumes *j1* < *k* **and** *j2* < *k* **and** *length tps* = *k* **and** *rneigh* (*tps* ! *j1*) *H n* **and** *t* ≤ *n*
shows *execute* (*tm-trans-until j1 j2 H f*) (*0*, *tps*) *t* =
(*0*, *tps*[*j1* := *tps* ! *j1* |+| *t*, *j2* := *transplant* (*tps* ! *j1*) (*tps* ! *j2*) *f t*])
using *assms*(*5*)

```
proof (induction t)
  case 0
  then show ?case
    using transplant-0 by simp
next
  case (Suc t)
  then have t < n by simp
  let ?M = tm-trans-until j1 j2 H f
  have execute ?M (0, tps) (Suc t) = exe ?M (execute ?M (0, tps) t)
    by simp
  also have ... = exe ?M (0, tps[j1 := tps ! j1 |+| t, j2 := transplant (tps ! j1) (tps ! j2) f t])
      (is - = exe ?M (0, ?tps))
    using Suc by simp
  also have ... = (0, ?tps[j1 := ?tps ! j1 |+| 1, j2 := ?tps ! j2 |:=| (f (?tps :.: j1)) |+| 1])
  proof (rule exe-tm-trans-until-2[where ?k=k])
    show j1 < k
      using assms(1) .
    show length (tps[j1 := tps ! j1 |+| t, j2 := transplant (tps ! j1) (tps ! j2) f t]) = k
      using assms by simp
    show (0, tps[j1 := tps ! j1 |+| t, j2 := transplant (tps ! j1) (tps ! j2) f t]) <.> j1 ∉ H
      using assms transplant-def rneigh-def ‹t < n›
      by (smt (verit) fst-conv length-list-update less-not-refl2 nth-list-update-eq nth-list-update-neq snd-conv)
  qed
  finally show ?case
    using assms transplant-upd
    by auto
    (smt (verit) add.commute fst-conv transplant-def transplant-upd less-not-refl2 list-update-overwrite list-update-swap
       nth-list-update-eq nth-list-update-neq plus-1-eq-Suc snd-conv)
qed

lemma execute-tm-ltrans-until-less:
  assumes j1 < k and j2 < k and length tps = k
    and lneigh (tps ! j1) H n
    and t ≤ n
    and n ≤ tps :#: j1
    and n ≤ tps :#: j2
  shows execute (tm-ltrans-until j1 j2 H f) (0, tps) t =
    (0, tps[j1 := tps ! j1 |−| t, j2 := ltransplant (tps ! j1) (tps ! j2) f t])
  using assms(5)
proof (induction t)
  case 0
  then show ?case
    using ltransplant-0 by simp
next
  case (Suc t)
  then have t < n
    by simp
  have 1: t < tps :#: j2
    using assms(7) Suc by simp
  have 2: t < tps :#: j1
    using assms(6) Suc by simp
  let ?M = tm-ltrans-until j1 j2 H f
  define tps' where tps' = tps[j1 := tps ! j1 |−| t, j2 := ltransplant (tps ! j1) (tps ! j2) f t]
  have execute ?M (0, tps) (Suc t) = exe ?M (execute ?M (0, tps) t)
    by simp
  also have ... = exe ?M (0, tps')
    using Suc tps'-def by simp
  also have ... = (0, tps'[j1 := tps' ! j1 |−| 1, j2 := tps' ! j2 |:=| (f (tps' :.: j1)) |−| 1])
  proof (rule exe-tm-ltrans-until-2[where ?k=k])
    show j1 < k
      using assms(1) .
    show length tps' = k
      using assms tps'-def by simp
```

```
    show (0, tps') <.> j1 ∉ H
      using assms ltransplant-def tps'-def lneigh-def ‹t < n›
      by (smt (verit) fst-conv length-list-update less-not-refl2 nth-list-update-eq nth-list-update-neq snd-conv)
  qed
  finally show ?case
    using tapes-ltransplant-upd[OF 1 2 assms(1,2,3) tps'-def] by simp
qed

lemma execute-tm-trans-until:
  assumes j1 < k and j2 < k and length tps = k and rneigh (tps ! j1) H n
  shows execute (tm-trans-until j1 j2 H f) (0, tps) (Suc n) =
    (1, tps[j1 := tps ! j1 |+| n, j2 := transplant (tps ! j1) (tps ! j2) f n])
proof -
  let ?M = tm-trans-until j1 j2 H f
  have execute ?M (0, tps) (Suc n) = exe ?M (execute ?M (0, tps) n)
    by simp
  also have ... = exe ?M (0, tps[j1 := tps ! j1 |+| n, j2 := transplant (tps ! j1) (tps ! j2) f n])
    using execute-tm-trans-until-less[where ?t=n] assms by simp
  also have ... = (1, tps[j1 := tps ! j1 |+| n, j2 := transplant (tps ! j1) (tps ! j2) f n])
      (is - = (1, ?tps))
  proof -
    have length ?tps = k
      using assms(3) by simp
    moreover have (0, ?tps) <.> j1 ∈ H
      using rneigh-def transplant-def assms
      by (smt (verit) fst-conv length-list-update less-not-refl2 nth-list-update-eq nth-list-update-neq snd-conv)
    ultimately show ?thesis
      using exe-tm-trans-until-1 assms by simp
  qed
  finally show ?thesis by simp
qed

lemma execute-tm-ltrans-until:
  assumes j1 < k and j2 < k and length tps = k
    and lneigh (tps ! j1) H n
    and n ≤ tps :#: j1
    and n ≤ tps :#: j2
  shows execute (tm-ltrans-until j1 j2 H f) (0, tps) (Suc n) =
    (1, tps[j1 := tps ! j1 |-| n, j2 := ltransplant (tps ! j1) (tps ! j2) f n])
proof -
  let ?M = tm-ltrans-until j1 j2 H f
  have execute ?M (0, tps) (Suc n) = exe ?M (execute ?M (0, tps) n)
    by simp
  also have ... = exe ?M (0, tps[j1 := tps ! j1 |-| n, j2 := ltransplant (tps ! j1) (tps ! j2) f n])
    using execute-tm-ltrans-until-less[where ?t=n] assms by simp
  also have ... = (1, tps[j1 := tps ! j1 |-| n, j2 := ltransplant (tps ! j1) (tps ! j2) f n])
      (is - = (1, ?tps))
  proof -
    have length ?tps = k
      using assms(3) by simp
    moreover have (0, ?tps) <.> j1 ∈ H
      using lneigh-def ltransplant-def assms
       by (smt (verit, ccfv-threshold) fst-conv length-list-update less-not-refl nth-list-update-eq nth-list-update-neq
snd-conv)
    ultimately show ?thesis
      using exe-tm-ltrans-until-1 assms by simp
  qed
  finally show ?thesis by simp
qed

lemma transits-tm-trans-until:
  assumes j1 < k and j2 < k and length tps = k and rneigh (tps ! j1) H n
  shows transits (tm-trans-until j1 j2 H f)
```

$(0, tps)$
$(Suc\ n)$
$(1, tps[j1 := tps ! j1 |+| n, j2 := transplant (tps ! j1) (tps ! j2) f n])$
using *execute-tm-trans-until*[*OF assms*] *transitsI*[*of - - Suc n - Suc n*] **by** *blast*

lemma *transits-tm-ltrans-until*:
assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* $= k$
and *lneigh* $(tps ! j1)$ *H n*
and $n \leq tps$:#: *j1*
and $n \leq tps$:#: *j2*
shows *transits* (*tm-ltrans-until j1 j2 H f*)
$(0, tps)$
$(Suc\ n)$
$(1, tps[j1 := tps ! j1 |-| n, j2 := ltransplant (tps ! j1) (tps ! j2) f n])$
using *execute-tm-ltrans-until*[*OF assms*] *transitsI*[*of - - Suc n - Suc n*] **by** *blast*

lemma *transforms-tm-trans-until*:
assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* $= k$ **and** *rneigh* $(tps ! j1)$ *H n*
shows *transforms* (*tm-trans-until j1 j2 H f*)
tps
$(Suc\ n)$
$(tps[j1 := tps ! j1 |+| n, j2 := transplant (tps ! j1) (tps ! j2) f n])$
using *tm-trans-until-def transforms-def transits-tm-trans-until*[*OF assms*] **by** *simp*

lemma *transforms-tm-ltrans-until*:
assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* $= k$
and *lneigh* $(tps ! j1)$ *H n*
and $n \leq tps$:#: *j1*
and $n \leq tps$:#: *j2*
shows *transforms* (*tm-ltrans-until j1 j2 H f*)
tps
$(Suc\ n)$
$(tps[j1 := tps ! j1 |-| n, j2 := ltransplant (tps ! j1) (tps ! j2) f n])$
using *tm-ltrans-until-def transforms-def transits-tm-ltrans-until*[*OF assms*] **by** *simp*

corollary *transforms-tm-trans-untilI* [*transforms-intros*]:
assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* $= k$
and *rneigh* $(tps ! j1)$ *H n*
and $t = Suc\ n$
and $tps' = tps[j1 := tps ! j1 |+| n, j2 := transplant (tps ! j1) (tps ! j2) f n]$
shows *transforms* (*tm-trans-until j1 j2 H f*) *tps t tps'*
using *transforms-tm-trans-until*[*OF assms*(*1*−*4*)] *assms*(*5*,*6*) **by** *simp*

corollary *transforms-tm-ltrans-untilI* [*transforms-intros*]:
assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* $= k$
and *lneigh* $(tps ! j1)$ *H n*
and $n \leq tps$:#: *j1*
and $n \leq tps$:#: *j2*
and $t = Suc\ n$
and $tps' = tps[j1 := tps ! j1 |-| n, j2 := ltransplant (tps ! j1) (tps ! j2) f n]$
shows *transforms* (*tm-ltrans-until j1 j2 H f*) *tps t tps'*
using *transforms-tm-ltrans-until*[*OF assms*(*1*−*6*)] *assms*(*7*,*8*) **by** *simp*

Copying one tape to another

If we set the symbol map *f* in *tm-trans-until* to the identity function, we get a Turing machine that simply makes a copy.

definition *tm-cp-until* :: *tapeidx* ⇒ *tapeidx* ⇒ *symbol set* ⇒ *machine* **where**
tm-cp-until j1 j2 H ≡ *tm-trans-until j1 j2 H id*

lemma *id-symbol*: $\forall h<G.\ (id :: symbol \Rightarrow symbol)\ h < G$
by *simp*

lemma *tm-cp-until-tm*:
  assumes $0 < j2$ **and** $j1 < k$ **and** $j2 < k$ **and** $G \geq 4$
  shows *turing-machine k G* (*tm-cp-until j1 j2 H*)
  unfolding *tm-cp-until-def* **using** *tm-trans-until-tm id-symbol assms turing-machine-def* **by** *simp*

abbreviation *implant* :: *tape* $\Rightarrow$ *tape* $\Rightarrow$ *nat* $\Rightarrow$ *tape* **where**
  implant tp1 tp2 n $\equiv$ *transplant tp1 tp2 id n*

lemma *implant*: *implant tp1 tp2 n* =
  (λi. *if snd tp2* $\leq i \wedge i <$ *snd tp2* + *n then fst tp1* (*snd tp1* + *i* − *snd tp2*) *else fst tp2 i*,
   snd tp2 + *n*)
  using *transplant-def* **by** *auto*

lemma *implantI*:
  assumes *tp′* =
  (λi. *if snd tp2* $\leq i \wedge i <$ *snd tp2* + *n then fst tp1* (*snd tp1* + *i* − *snd tp2*) *else fst tp2 i*,
   snd tp2 + *n*)
  shows *implant tp1 tp2 n* = *tp′*
  using *implant assms* **by** *simp*

lemma *fst-snd-pair*: *fst t* = *a* $\Longrightarrow$ *snd t* = *b* $\Longrightarrow$ *t* = (*a*, *b*)
  by *auto*

lemma *implantI′*:
  assumes *fst tp′* =
  (λi. *if snd tp2* $\leq i \wedge i <$ *snd tp2* + *n then fst tp1* (*snd tp1* + *i* − *snd tp2*) *else fst tp2 i*)
   and *snd tp′* = *snd tp2* + *n*
  shows *implant tp1 tp2 n* = *tp′*
  using *implantI fst-snd-pair*[*OF assms*] **by** *simp*

lemma *implantI′′*:
  assumes $\bigwedge i$. *snd tp2* $\leq i \wedge i <$ *snd tp2* + *n* $\Longrightarrow$ *fst tp′ i* = *fst tp1* (*snd tp1* + *i* − *snd tp2*)
   and $\bigwedge i$. *i* < *snd tp2* $\Longrightarrow$ *fst tp′ i* = *fst tp2 i*
   and $\bigwedge i$. *snd tp2* + *n* $\leq i \Longrightarrow$ *fst tp′ i* = *fst tp2 i*
  assumes *snd tp′* = *snd tp2* + *n*
  shows *implant tp1 tp2 n* = *tp′*
  using *assms implantI′* **by** (*meson linorder-le-less-linear*)

lemma *implantI′′′*:
  assumes $\bigwedge i$. *i2* $\leq i \wedge i <$ *i2* + *n* $\Longrightarrow$ *ys i* = *ys1* (*i1* + *i* − *i2*)
   and $\bigwedge i$. *i* < *i2* $\Longrightarrow$ *ys i* = *ys2 i*
   and $\bigwedge i$. *i2* + *n* $\leq i \Longrightarrow$ *ys i* = *ys2 i*
  assumes *i* = *i2* + *n*
  shows *implant* (*ys1*, *i1*) (*ys2*, *i2*) *n* = (*ys*, *i*)
  using *assms implantI′′* **by** *auto*

lemma *implant-self*: *implant tp tp n* = *tp* |+| *n*
  unfolding *transplant-def* **by** *auto*

lemma *transforms-tm-cp-untilI* [*transforms-intros*]:
  assumes *j1* < *k* **and** *j2* < *k* **and** *length tps* = *k*
   and *rneigh* (*tps* ! *j1*) *H n*
   and *t* = *Suc n*
   and *tps′* = *tps*[*j1* := *tps* ! *j1* |+| *n*, *j2* := *implant* (*tps* ! *j1*) (*tps* ! *j2*) *n*]
  shows *transforms* (*tm-cp-until j1 j2 H*) *tps t tps′*
  unfolding *tm-cp-until-def* **using** *transforms-tm-trans-untilI*[*OF assms*(*1*−*6*)] **by** *simp*

lemma *implant-contents*:
  assumes *i* > *0* **and** *n* + (*i* − *1*) $\leq$ *length xs*
  shows *implant* ($\lfloor$*xs*$\rfloor$, *i*) ($\lfloor$*ys*$\rfloor$, *Suc* (*length ys*)) *n* =
   ($\lfloor$*ys* @ (*take n* (*drop* (*i* − *1*) *xs*))$\rfloor$, *Suc* (*length ys*) + *n*)
   (**is** *?lhs* = *?rhs*)
proof −

```
  have lhs: ?lhs =
   (λj. if Suc (length ys) ≤ j ∧ j < Suc (length ys) + n then ⌊xs⌋ (i + j − Suc (length ys)) else ⌊ys⌋ j,
    Suc (length ys) + n)
    using implant by auto
  let ?zs = ys @ (take n (drop (i − 1) xs))
  have lenzs: length ?zs = length ys + n
    using assms by simp
  have fst-rhs: fst ?rhs = (λj. if j = 0 then 1 else if j ≤ length ys + n then ?zs ! (j − 1) else 0)
    using assms by auto

  have (λj. if Suc (length ys) ≤ j ∧ j < Suc (length ys) + n then ⌊xs⌋ (i + j − Suc (length ys)) else ⌊ys⌋ j) =
      (λj. if j = 0 then 1 else if j ≤ length ys + n then ?zs ! (j − 1) else 0)
    (is ?l = ?r)
  proof
    fix j
    consider
        j = 0
      | j > 0 ∧ j ≤ length ys
      | j > length ys ∧ j ≤ length ys + n
      | j > length ys + n
      by linarith
    then show ?l j = ?r j
    proof (cases)
      case 1
      then show ?thesis
        by simp
    next
      case 2
      then show ?thesis
          using assms contents-def by (smt (verit) Suc-diff-1 less-trans-Suc not-add-less1 not-le not-less-eq-eq
nth-append)
    next
      case 3
      then have ?r j = ?zs ! (j − 1)
        by simp
      also have ... = take n (drop (i − 1) xs) ! (j − 1 − length ys)
        using 3 lenzs
          by (metis add.right-neutral diff-is-0-eq le-add-diff-inverse not-add-less2 not-le not-less-eq nth-append
plus-1-eq-Suc)
      also have ... = take n (drop (i − 1) xs) ! (j − Suc (length ys))
        by simp
      also have ... = xs ! (i − 1 + j − Suc (length ys))
        using 3 assms by auto
      also have ... = ⌊xs⌋ (i + j − Suc (length ys))
        using assms contents-def 3 by auto
      also have ... = ?l j
        using 3 by simp
      finally have ?r j = ?l j .
      then show ?thesis
        by simp
    next
      case 4
      then show ?thesis
        by simp
    qed
  qed
  then show ?thesis
    using lhs fst-rhs by simp
qed
```

Moving to the next specific symbol

Copying a tape to itself means just moving to the right.

definition *tm-right-until* :: *tapeidx* ⇒ *symbol set* ⇒ *machine* **where**
tm-right-until j H ≡ *tm-cp-until j j H*

Copying a tape to itself does not change the tape. So this is a Turing machine even for the input tape $j = 0$, unlike *tm-cp-until* where the target tape cannot, in general, be the input tape.

lemma *tm-right-until-tm*:
assumes $j < k$ **and** $k \geq 2$ **and** $G \geq 4$
shows *turing-machine k G* (*tm-right-until j H*)
unfolding *tm-right-until-def tm-cp-until-def tm-trans-until-def cmd-trans-until-def*
using *assms turing-machine-def*
by *auto*

lemma *transforms-tm-right-untilI* [*transforms-intros*]:
assumes $j <$ *length tps*
and *rneigh* (*tps* ! *j*) *H n*
and $t =$ *Suc n*
and *tps'* = (*tps*[*j* := *tps* ! *j* |+| *n*])
shows *transforms* (*tm-right-until j H*) *tps t tps'*
using *transforms-tm-cp-untilI assms implant-self tm-right-until-def*
by (*metis list-update-id nth-list-update-eq*)

Translating to a constant symbol

Another way to specialize *tm-trans-until* and *tm-ltrans-until* is to have a constant function f.

definition *tm-const-until* :: *tapeidx* ⇒ *tapeidx* ⇒ *symbol set* ⇒ *symbol* ⇒ *machine* **where**
tm-const-until j1 j2 H h ≡ *tm-trans-until j1 j2 H* (λ-. *h*)

lemma *tm-const-until-tm*:
assumes $0 < j2$ **and** $j1 < k$ **and** $j2 < k$ **and** $h < G$ **and** $k \geq 2$ **and** $G \geq 4$
shows *turing-machine k G* (*tm-const-until j1 j2 H h*)
unfolding *tm-const-until-def* **using** *tm-trans-until-tm assms turing-machine-def* **by** *metis*

Continuing with our fantasy names ending in *-plant*, we name the operation *constplant*.

abbreviation *constplant* :: *tape* ⇒ *symbol* ⇒ *nat* ⇒ *tape* **where**
constplant tp2 h n ≡ *transplant* (λ-. *0*, *0*) *tp2* (λ-. *h*) *n*

lemma *constplant-transplant*: *constplant tp2 h n* = *transplant tp1 tp2* (λ-. *h*) *n*
using *transplant-def* **by** *simp*

lemma *constplant*: *constplant tp2 h n* =
(λ*i*. *if snd tp2* ≤ *i* ∧ *i* < *snd tp2* + *n then h else fst tp2 i*,
snd tp2 + *n*)
using *transplant-def* **by** *simp*

lemma *transforms-tm-const-untilI* [*transforms-intros*]:
assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* = *k*
and *rneigh* (*tps* ! *j1*) *H n*
and $t =$ *Suc n*
and *tps'* = *tps*[*j1* := *tps* ! *j1* |+| *n*, *j2* := *constplant* (*tps* ! *j2*) *h n*]
shows *transforms* (*tm-const-until j1 j2 H h*) *tps t tps'*
unfolding *tm-const-until-def* **using** *transforms-tm-trans-untilI assms constplant-transplant* **by** *metis*

definition *tm-lconst-until* :: *tapeidx* ⇒ *tapeidx* ⇒ *symbol set* ⇒ *symbol* ⇒ *machine* **where**
tm-lconst-until j1 j2 H h ≡ *tm-ltrans-until j1 j2 H* (λ-. *h*)

lemma *tm-lconst-until-tm*:
assumes $0 < j2$ **and** $j1 < k$ **and** $j2 < k$ **and** $h < G$ **and** $k \geq 2$ **and** $G \geq 4$
shows *turing-machine k G* (*tm-lconst-until j1 j2 H h*)
unfolding *tm-lconst-until-def* **using** *tm-ltrans-until-tm assms turing-machine-def* **by** *metis*

abbreviation *lconstplant* :: *tape* ⇒ *symbol* ⇒ *nat* ⇒ *tape* **where**
lconstplant tp2 h n ≡ *ltransplant* (λ-. *0*, *0*) *tp2* (λ-. *h*) *n*

lemma *lconstplant-ltransplant*: *lconstplant tp2 h n* = *ltransplant tp1 tp2* (λ-. *h*) *n*
using *ltransplant-def* **by** *simp*

lemma *lconstplant*: *lconstplant tp2 h n* =
(λ*i*. *if snd tp2* − *n* < *i* ∧ *i* ≤ *snd tp2 then h else fst tp2 i*,
snd tp2 − *n*)
using *ltransplant-def* **by** *simp*

lemma *transforms-tm-lconst-untilI* [*transforms-intros*]:
assumes *0* < *j2* **and** *j1* < *k* **and** *j2* < *k* **and** *length tps* = *k*
and *lneigh* (*tps* ! *j1*) *H n*
and *n* ≤ *tps* :#: *j1*
and *n* ≤ *tps* :#: *j2*
and *t* = *Suc n*
and *tps'* = *tps*[*j1* := *tps* ! *j1* |−| *n*, *j2* := *lconstplant* (*tps* ! *j2*) *h n*]
shows *transforms* (*tm-lconst-until j1 j2 H h*) *tps t tps'*
unfolding *tm-lconst-until-def* **using** *transforms-tm-ltrans-untilI assms lconstplant-ltransplant* **by** *metis*

2.4.4 Writing single symbols

The next command writes a fixed symbol h to tape j. It does not move a tape head.

definition *cmd-write* :: *tapeidx* ⇒ *symbol* ⇒ *command* **where**
cmd-write j h rs ≡ (*1*, *map* (λ*i*. (*if i* = *j then h else rs* ! *i*, *Stay*)) [*0*..<*length rs*])

lemma *sem-cmd-write*: *sem* (*cmd-write j h*) (*0*, *tps*) = (*1*, *tps*[*j* := *tps* ! *j* |:=| *h*])
using *cmd-write-def read-length act-Stay* **by** (*intro semI*) *auto*

definition *tm-write* :: *tapeidx* ⇒ *symbol* ⇒ *machine* **where**
tm-write j h ≡ [*cmd-write j h*]

lemma *tm-write-tm*:
assumes *0* < *j* **and** *j* < *k* **and** *h* < *G* **and** *G* ≥ *4*
shows *turing-machine k G* (*tm-write j h*)
unfolding *tm-write-def cmd-write-def* **using** *assms turing-machine-def* **by** *auto*

lemma *transforms-tm-writeI* [*transforms-intros*]:
assumes *tps'* = *tps*[*j* := *tps* ! *j* |:=| *h*]
shows *transforms* (*tm-write j h*) *tps 1 tps'*
unfolding *tm-write-def*
using *sem-cmd-write exe-lt-length assms tm-write-def transits-def transforms-def*
by *auto*

The next command writes the symbol to tape j_2 that results from applying a function f to the symbol read from tape j_1. It does not move any tape heads.

definition *cmd-trans2* :: *tapeidx* ⇒ *tapeidx* ⇒ (*symbol* ⇒ *symbol*) ⇒ *command* **where**
cmd-trans2 j1 j2 f rs ≡ (*1*, *map* (λ*i*. (*if i* = *j2 then f* (*rs* ! *j1*) *else rs* ! *i*, *Stay*)) [*0*..<*length rs*])

lemma *sem-cmd-trans2*:
assumes *j1* < *length tps*
shows *sem* (*cmd-trans2 j1 j2 f*) (*0*, *tps*) = (*1*, *tps*[*j2* := *tps* ! *j2* |:=| (*f* (*tps* :.: *j1*))])
using *cmd-trans2-def tapes-at-read assms read-length act-Stay* **by** (*intro semI*) *auto*

definition *tm-trans2* :: *tapeidx* ⇒ *tapeidx* ⇒ (*symbol* ⇒ *symbol*) ⇒ *machine* **where**
tm-trans2 j1 j2 f ≡ [*cmd-trans2 j1 j2 f*]

lemma *tm-trans2-tm*:
assumes *j1* < *k* **and** *0* < *j2* **and** *j2* < *k* **and** ∀ *h* < *G*. *f h* < *G* **and** *k* ≥ *2* **and** *G* ≥ *4*
shows *turing-machine k G* (*tm-trans2 j1 j2 f*)
unfolding *tm-trans2-def cmd-trans2-def* **using** *assms turing-machine-def* **by** *auto*

lemma *exe-tm-trans2*:
assumes *j1* < *length tps*

shows *exe* (*tm-trans2 j1 j2 f*) (*0*, *tps*) = (*1*, *tps*[*j2* := *tps* ! *j2* |:=| (*f* (*tps* :.: *j1*))])
unfolding *tm-trans2-def* **using** *sem-cmd-trans2 assms exe-lt-length* **by** *simp*

lemma *execute-tm-trans2*:
assumes *j1* < *length tps*
shows *execute* (*tm-trans2 j1 j2 f*) (*0*, *tps*) *1* = (*1*, *tps*[*j2* := *tps* ! *j2* |:=| (*f* (*tps* :.: *j1*))])
using *assms exe-tm-trans2* **by** *simp*

lemma *transits-tm-trans2*:
assumes *j1* < *length tps*
shows *transits* (*tm-trans2 j1 j2 f*) (*0*, *tps*) *1* (*1*, *tps*[*j2* := *tps* ! *j2* |:=| (*f* (*tps* :.: *j1*))])
using *assms execute-tm-trans2 transits-def* **by** *auto*

lemma *transforms-tm-trans2*:
assumes *j1* < *length tps*
shows *transforms* (*tm-trans2 j1 j2 f*) *tps 1* (*tps*[*j2* := *tps* ! *j2* |:=| (*f* (*tps* :.: *j1*))])
using *tm-trans2-def assms transits-tm-trans2 transforms-def* **by** *simp*

lemma *transforms-tm-trans2I* [*transforms-intros*]:
assumes *j1* < *length tps* **and** *tps'* = *tps*[*j2* := *tps* ! *j2* |:=| (*f* (*tps* :.: *j1*))]
shows *transforms* (*tm-trans2 j1 j2 f*) *tps 1 tps'*
using *assms transforms-tm-trans2* **by** *simp*

Equating the two tapes in *tm-trans2*, we can map a symbol in-place.

definition *tm-trans* :: *tapeidx* ⇒ (*symbol* ⇒ *symbol*) ⇒ *machine* **where**
tm-trans j f ≡ *tm-trans2 j j f*

lemma *tm-trans-tm*:
assumes *0* < *j* **and** *j* < *k* **and** ∀ *h* < *G*. *f h* < *G* **and** *G* ≥ *4*
shows *turing-machine k G* (*tm-trans j f*)
unfolding *tm-trans-def* **using** *tm-trans2-tm assms* **by** *simp*

lemma *transforms-tm-transI* [*transforms-intros*]:
assumes *j* < *length tps* **and** *tps'* = *tps*[*j* := *tps* ! *j* |:=| (*f* (*tps* :.: *j*))]
shows *transforms* (*tm-trans j f*) *tps 1 tps'*
using *assms transforms-tm-trans2 tm-trans-def* **by** *simp*

The next command is like the previous one, except it also moves the tape head to the right.

definition *cmd-rtrans* :: *tapeidx* ⇒ (*symbol* ⇒ *symbol*) ⇒ *command* **where**
cmd-rtrans j f rs ≡ (*1*, *map* (λ*i*. (*if i* = *j then f* (*rs* ! *i*) *else rs* ! *i*, *if i* = *j then Right else Stay*)) [*0*..<*length rs*])

lemma *sem-cmd-rtrans*:
assumes *j* < *length tps*
shows *sem* (*cmd-rtrans j f*) (*0*, *tps*) = (*1*, *tps*[*j* := *tps* ! *j* |:=| (*f* (*tps* :.: *j*)) |+| *1*])
using *cmd-rtrans-def tapes-at-read read-length assms act-Stay act-Right*
by (*intro semI*) *auto*

definition *tm-rtrans* :: *tapeidx* ⇒ (*symbol* ⇒ *symbol*) ⇒ *machine* **where**
tm-rtrans j f ≡ [*cmd-rtrans j f*]

lemma *tm-rtrans-tm*:
assumes *0* < *j* **and** *j* < *k* **and** ∀ *h*<*G*. *f h* < *G* **and** *k* ≥ *2* **and** *G* ≥ *4*
shows *turing-machine k G* (*tm-rtrans j f*)
unfolding *tm-rtrans-def cmd-rtrans-def* **using** *assms turing-machine-def* **by** *auto*

lemma *exe-tm-rtrans*:
assumes *j* < *length tps*
shows *exe* (*tm-rtrans j f*) (*0*, *tps*) = (*1*, *tps*[*j* := *tps* ! *j* |:=| (*f* (*tps* :.: *j*)) |+| *1*])
unfolding *tm-rtrans-def* **using** *sem-cmd-rtrans assms exe-lt-length* **by** *simp*

lemma *execute-tm-rtrans*:
assumes *j* < *length tps*

shows *execute* (*tm-rtrans j f*) (*0*, *tps*) *1* = (*1*, *tps*[*j* := *tps* ! *j* |:=| (*f* (*tps* :.: *j*)) |+| *1*])
using *assms exe-tm-rtrans* **by** *simp*

lemma *transits-tm-rtrans*:
assumes *j* < *length tps*
shows *transits* (*tm-rtrans j f*) (*0*, *tps*) *1* (*1*, *tps*[*j* := *tps* ! *j* |:=| (*f* (*tps* :.: *j*)) |+| *1*])
using *assms execute-tm-rtrans transits-def* **by** *auto*

lemma *transforms-tm-rtrans*:
assumes *j* < *length tps*
shows *transforms* (*tm-rtrans j f*) *tps 1* (*tps*[*j* := *tps* ! *j* |:=| (*f* (*tps* :.: *j*)) |+| *1*])
using *assms transits-tm-rtrans transforms-def* **by** (*metis One-nat-def length-Cons list.size*(*3*) *tm-rtrans-def*)

lemma *transforms-tm-rtransI* [*transforms-intros*]:
assumes *j* < *length tps* **and** *tps'* = *tps*[*j* := *tps* ! *j* |:=| (*f* (*tps* :.: *j*)) |+| *1*]
shows *transforms* (*tm-rtrans j f*) *tps 1 tps'*
using *assms transforms-tm-rtrans* **by** *simp*

The next command writes a fixed symbol h to all tapes in the set J.

definition *cmd-write-many* :: *tapeidx set* ⇒ *symbol* ⇒ *command* **where**
cmd-write-many J h rs ≡ (*1*, *map* (λ*i*. (*if i* ∈ *J then h else rs* ! *i*, *Stay*)) [*0*..<*length rs*])

lemma *proper-cmd-write-many*: *proper-command k* (*cmd-write-many J h*)
unfolding *cmd-write-many-def* **by** *auto*

lemma *sem-cmd-write-many*:
shows *sem* (*cmd-write-many J h*) (*0*, *tps*) =
(*1*, *map* (λ*j*. *if j* ∈ *J then tps* ! *j* |:=| *h else tps* ! *j*) [*0*..<*length tps*])
using *cmd-write-many-def read-length act-Stay*
by (*intro semI*[*OF proper-cmd-write-many*]) *auto*

definition *tm-write-many* :: *tapeidx set* ⇒ *symbol* ⇒ *machine* **where**
tm-write-many J h ≡ [*cmd-write-many J h*]

lemma *tm-write-many-tm*:
assumes *0* ∉ *J* **and** ∀ *j*∈*J*. *j* < *k* **and** *h* < *G* **and** *k* ≥ *2* **and** *G* ≥ *4*
shows *turing-machine k G* (*tm-write-many J h*)
unfolding *tm-write-many-def cmd-write-many-def* **using** *assms turing-machine-def* **by** *auto*

lemma *exe-tm-write-many*: *exe* (*tm-write-many J h*) (*0*, *tps*) =
(*1*, *map* (λ*j*. *if j* ∈ *J then tps* ! *j* |:=| *h else tps* ! *j*) [*0*..<*length tps*])
unfolding *tm-write-many-def* **using** *sem-cmd-write-many exe-lt-length* **by** *simp*

lemma *execute-tm-write-many*: *execute* (*tm-write-many J h*) (*0*, *tps*) *1* =
(*1*, *map* (λ*j*. *if j* ∈ *J then tps* ! *j* |:=| *h else tps* ! *j*) [*0*..<*length tps*])
using *exe-tm-write-many* **by** *simp*

lemma *transforms-tm-write-many*:
transforms (*tm-write-many J h*) *tps 1* (*map* (λ*j*. *if j* ∈ *J then tps* ! *j* |:=| *h else tps* ! *j*) [*0*..<*length tps*])
using *execute-tm-write-many transits-def transforms-def tm-write-many-def* **by** *auto*

lemma *transforms-tm-write-manyI* [*transforms-intros*]:
assumes ∀ *j*∈*J*. *j* < *k*
and *length tps* = *k*
and *length tps'* = *k*
and ⋀*j*. *j* ∈ *J* ⟹ *tps'* ! *j* = *tps* ! *j* |:=| *h*
and ⋀*j*. *j* < *k* ⟹ *j* ∉ *J* ⟹ *tps'* ! *j* = *tps* ! *j*
shows *transforms* (*tm-write-many J h*) *tps 1 tps'*
proof −
have *tps'* = *map* (λ*j*. *if j* ∈ *J then tps* ! *j* |:=| *h else tps* ! *j*) [*0*..<*length tps*]
using *assms* **by** (*intro nth-equalityI*) *simp-all*
then show *?thesis*
using *assms transforms-tm-write-many* **by** *auto*

qed

2.4.5 Writing a symbol multiple times

In this section we devise a Turing machine that writes the symbol sequence h^m with a hard-coded symbol h and number m to a tape. The resulting tape is defined by the next operation:

definition *overwrite* :: *tape* ⇒ *symbol* ⇒ *nat* ⇒ *tape* **where**
  overwrite tp h m ≡ (*λi. if snd tp* ≤ *i* ∧ *i* < *snd tp* + *m then h else fst tp i*, *snd tp* + *m*)

lemma *overwrite-0*: *overwrite tp h 0* = *tp*
proof −
  have *snd* (*overwrite tp h 0*) = *snd tp*
    unfolding *overwrite-def* **by** *simp*
  moreover have *fst* (*overwrite tp h 0*) = *fst tp*
    unfolding *overwrite-def* **by** *auto*
  ultimately show *?thesis*
    using *prod-eqI* **by** *blast*
qed

lemma *overwrite-upd*: (*overwrite tp h t*) |:=| *h* |+| *1* = *overwrite tp h* (*Suc t*)
  using *overwrite-def* **by** *auto*

lemma *overwrite-upd′*:
  assumes *j* < *length tps* **and** *tps′* = *tps*[*j* := *overwrite* (*tps* ! *j*) *h t*]
  shows (*tps*[*j* := *overwrite* (*tps* ! *j*) *h t*])[*j* := *tps′* ! *j* |:=| *h* |+| *1*] =
    tps[*j* := *overwrite* (*tps* ! *j*) *h* (*Suc t*)]
  using *assms overwrite-upd* **by** *simp*

The next command writes the symbol h to the tape j and moves the tape head to the right.

definition *cmd-char* :: *tapeidx* ⇒ *symbol* ⇒ *command* **where**
  cmd-char j z = *cmd-rtrans j* (*λ-. z*)

lemma *turing-command-char*:
  assumes *0* < *j* **and** *j* < *k* **and** *h* < *G*
  shows *turing-command k 1 G* (*cmd-char j h*)
  unfolding *cmd-char-def cmd-rtrans-def* **using** *assms* **by** *auto*

definition *tm-char* :: *tapeidx* ⇒ *symbol* ⇒ *machine* **where**
  tm-char j z ≡ [*cmd-char j z*]

lemma *tm-char-tm*:
  assumes *0* < *j* **and** *j* < *k* **and** *G* ≥ *4* **and** *z* < *G*
  shows *turing-machine k G* (*tm-char j z*)
  using *assms turing-command-char tm-char-def* **by** *auto*

lemma *transforms-tm-charI* [*transforms-intros*]:
  assumes *j* < *length tps* **and** *tps′* = *tps*[*j* := *tps* ! *j* |:=| *z* |+| *1*]
  shows *transforms* (*tm-char j z*) *tps 1 tps′*
  using *assms transforms-tm-rtransI tm-char-def cmd-char-def tm-rtrans-def* **by** *metis*

lemma *sem-cmd-char*:
  assumes *j* < *length tps*
  shows *sem* (*cmd-char j h*) (*0*, *tps*) = (*1*, *tps*[*j* := *tps* ! *j* |:=| *h* |+| *1*])
  using *cmd-char-def cmd-rtrans-def tapes-at-read read-length assms act-Right*
  by (*intro semI*) *auto*

The next TM is a sequence of m *cmd-char* commands properly relocated. It writes a sequence of m times the symbol h to tape j.

definition *tm-write-repeat* :: *tapeidx* ⇒ *symbol* ⇒ *nat* ⇒ *machine* **where**
  tm-write-repeat j h m ≡ *map* (*λi. relocate-cmd i* (*cmd-char j h*)) [*0*..<*m*]

lemma *tm-write-repeat-tm*:

```
  assumes 0 < j and j < k and h < G and k ≥ 2 and G ≥ 4
  shows turing-machine k G (tm-write-repeat j h m)
proof
  let ?M = tm-write-repeat j h m
  show 2 ≤ k and 4 ≤ G
    using assms(4,5) .
  show turing-command k (length ?M) G (?M ! i) if i < length ?M for i
  proof −
    have ?M ! i = relocate-cmd i (cmd-char j h)
      using that tm-write-repeat-def by simp
    then have turing-command k (1 + i) G (?M ! i)
      using assms turing-command-char turing-command-relocate-cmd by metis
    then show ?thesis
      using turing-command-mono that by simp
  qed
qed

lemma exe-tm-write-repeat:
  assumes j < length tps and q < m
  shows exe (tm-write-repeat j h m) (q, tps) = (Suc q, tps[j := tps ! j |:=| h |+| 1])
  using sem-cmd-char assms sem sem-relocate-cmd tm-write-repeat-def by (auto simp add: exe-lt-length)

lemma execute-tm-write-repeat:
  assumes j < length tps and t ≤ m
  shows execute (tm-write-repeat j h m) (0, tps) t = (t, tps[j := overwrite (tps ! j) h t])
  using assms(2)
proof (induction t)
  case 0
  then show ?case using overwrite-0 by simp
next
  case (Suc t)
  then have t < m by simp
  have execute (tm-write-repeat j h m) (0, tps) (Suc t) = exe (tm-write-repeat j h m) (execute (tm-write-repeat j
h m) (0, tps) t)
    by simp
  also have ... = exe (tm-write-repeat j h m) (t, tps[j := overwrite (tps ! j) h t])
    using Suc by simp
  also have ... = (Suc t, tps[j := overwrite (tps ! j) h (Suc t)])
    using ‹t < m› exe-tm-write-repeat assms overwrite-upd′ by simp
  finally show ?case .
qed

lemma transforms-tm-write-repeatI [transforms-intros]:
  assumes j < length tps and tps′ = tps[j := overwrite (tps ! j) h m]
  shows transforms (tm-write-repeat j h m) tps m tps′
  using assms execute-tm-write-repeat transits-def transforms-def tm-write-repeat-def by auto
```

2.4.6 Moving to the start of the tape

The next command moves the head on tape j to the left until it reaches a symbol from the set H:

```
definition cmd-left-until :: symbol set ⇒ tapeidx ⇒ command where
  cmd-left-until H j rs ≡
    if rs ! j ∈ H
    then (1, map (λr. (r, Stay)) rs)
    else (0, map (λi. (rs ! i, if i = j then Left else Stay)) [0..<length rs])

lemma sem-cmd-left-until-1:
  assumes j < k and length tps = k and (0, tps) <.> j ∈ H
  shows sem (cmd-left-until H j) (0, tps) = (1, tps)
  using cmd-left-until-def tapes-at-read read-length assms act-Stay
  by (intro semI) auto

lemma sem-cmd-left-until-2:
```

assumes *j* < *k* **and** *length tps* = *k* **and** (*0*, *tps*) <.> *j* ∉ *H*
shows *sem* (*cmd-left-until H j*) (*0*, *tps*) = (*0*, *tps*[*j* := *tps* ! *j* |−| *1*])
using *cmd-left-until-def tapes-at-read read-length assms act-Stay act-Left*
by (*intro semI*) *auto*

definition *tm-left-until* :: *symbol set* ⇒ *tapeidx* ⇒ *machine* **where**
tm-left-until H j ≡ [*cmd-left-until H j*]

lemma *tm-left-until-tm*:
assumes *k* ≥ *2* **and** *G* ≥ *4*
shows *turing-machine k G* (*tm-left-until H j*)
unfolding *tm-left-until-def cmd-left-until-def* **using** *assms turing-machine-def* **by** *auto*

A *begin tape* for a set of symbols has one of these symbols only in cell zero. It generalizes the concept of clean tapes, where the set of symbols is {▹}.

definition *begin-tape* :: *symbol set* ⇒ *tape* ⇒ *bool* **where**
begin-tape H tp ≡ ∀ *i*. *fst tp i* ∈ *H* ⟷ *i* = *0*

lemma *begin-tapeI*:
assumes *fst tp 0* ∈ *H* **and** ⋀*i*. *i* > *0* ⟹ *fst tp i* ∉ *H*
shows *begin-tape H tp*
unfolding *begin-tape-def* **using** *assms* **by** *auto*

lemma *exe-tm-left-until-1*:
assumes *j* < *length tps* **and** (*0*, *tps*) <.> *j* ∈ *H*
shows *exe* (*tm-left-until H j*) (*0*, *tps*) = (*1*, *tps*)
using *tm-left-until-def assms exe-lt-length sem-cmd-left-until-1* **by** *auto*

lemma *exe-tm-left-until-2*:
assumes *j* < *length tps* **and** (*0*, *tps*) <.> *j* ∉ *H*
shows *exe* (*tm-left-until H j*) (*0*, *tps*) = (*0*, *tps*[*j* := *tps* ! *j* |−| *1*])
using *tm-left-until-def assms exe-lt-length sem-cmd-left-until-2* **by** *auto*

We do not show the semantics of *tm-left-until* for the general case, but only for when applied to begin tapes.

lemma *execute-tm-left-until-less*:
assumes *j* < *length tps* **and** *begin-tape H* (*tps* ! *j*) **and** *t* ≤ *tps* :#: *j*
shows *execute* (*tm-left-until H j*) (*0*, *tps*) *t* = (*0*, *tps*[*j* := *tps* ! *j* |−| *t*])
using *assms*(*3*)
proof (*induction t*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc t*)
then have *fst* (*tps* ! *j*) (*snd* (*tps* ! *j*) − *t*) ∉ *H*
using *assms begin-tape-def* **by** *simp*
then have *neq-0*: *fst* (*tps* ! *j* |−| *t*) (*snd* (*tps* ! *j* |−| *t*)) ∉ *H*
by *simp*
have *execute* (*tm-left-until H j*) (*0*, *tps*) (*Suc t*) = *exe* (*tm-left-until H j*) (*execute* (*tm-left-until H j*) (*0*, *tps*) *t*)
by *simp*
also have ... = *exe* (*tm-left-until H j*) (*0*, *tps*[*j* := *tps* ! *j* |−| *t*])
using *Suc* **by** *simp*
also have ... = (*0*, *tps*[*j* := *tps* ! *j* |−| (*Suc t*)])
using *neq-0 exe-tm-left-until-2 assms* **by** *simp*
finally show *?case* **by** *simp*
qed

lemma *execute-tm-left-until*:
assumes *j* < *length tps* **and** *begin-tape H* (*tps* ! *j*)
shows *execute* (*tm-left-until H j*) (*0*, *tps*) (*Suc* (*tps* :#: *j*)) = (*1*, *tps*[*j* := *tps* ! *j* |#=| *0*])
using *assms begin-tape-def exe-tm-left-until-1 execute-tm-left-until-less* **by** *simp*

lemma *transits-tm-left-until*:

assumes $j < length\ tps$ **and** *begin-tape H* $(tps\ !\ j)$
shows *transits* (*tm-left-until H j*) $(0,\ tps)$ (*Suc* $(tps\ :\#:\ j)$) $(1,\ tps[j := tps\ !\ j\ |\#=|\ 0])$
using *execute-imp-transits*[*OF execute-tm-left-until*[*OF assms*]] **by** *simp*

lemma *transforms-tm-left-until*:
assumes $j < length\ tps$ **and** *begin-tape H* $(tps\ !\ j)$
shows *transforms* (*tm-left-until H j*) *tps* (*Suc* $(tps\ :\#:\ j)$) $(tps[j := tps\ !\ j\ |\#=|\ 0])$
using *tm-left-until-def transforms-def transits-tm-left-until*[*OF assms*] **by** *simp*

The most common case is $H = \{\triangleright\}$, which means the Turing machine moves the tape head left to the closest start symbol. On clean tapes it moves the tape head to the leftmost cell of the tape.

definition *tm-start* :: *tapeidx* $\Rightarrow$ *machine* **where**
tm-start $\equiv$ *tm-left-until* $\{1\}$

lemma *tm-start-tm*:
assumes $k \geq 2$ **and** $G \geq 4$
shows *turing-machine k G* (*tm-start j*)
unfolding *tm-start-def* **using** *assms tm-left-until-tm* **by** *simp*

lemma *transforms-tm-start*:
assumes $j < length\ tps$ **and** *clean-tape* $(tps\ !\ j)$
shows *transforms* (*tm-start j*) *tps* (*Suc* $(tps\ :\#:\ j)$) $(tps[j := tps\ !\ j\ |\#=|\ 0])$
using *tm-start-def assms transforms-tm-left-until begin-tape-def clean-tape-def* **by** (*metis insertCI singletonD*)

lemma *transforms-tm-startI* [*transforms-intros*]:
assumes $j < length\ tps$ **and** *clean-tape* $(tps\ !\ j)$
and $t = Suc\ (tps\ :\#:\ j)$
and $tps' = tps[j := tps\ !\ j\ |\#=|\ 0]$
shows *transforms* (*tm-start j*) *tps t tps'*
using *transforms-tm-start assms* **by** *simp*

The next Turing machine is the first instance in which we use the ;; operator with concrete Turing machines. It is also the first time we use the proof method *tform* for *transforms*. The TM performs a "carriage return" on a clean tape, that is, it moves to the first non-start symbol.

definition *tm-cr* :: *tapeidx* $\Rightarrow$ *machine* **where**
tm-cr j $\equiv$ *tm-start j* ;; *tm-right j*

lemma *tm-cr-tm*: $k \geq 2 \Longrightarrow G \geq 4 \Longrightarrow$ *turing-machine k G* (*tm-cr j*)
using *turing-machine-sequential-turing-machine* **by** (*simp add*: *tm-cr-def tm-right-tm tm-start-tm*)

lemma *transforms-tm-crI* [*transforms-intros*]:
assumes $j < length\ tps$
and *clean-tape* $(tps\ !\ j)$
and $t = tps\ :\#:\ j + 2$
and $tps' = tps[j := tps\ !\ j\ |\#=|\ 1]$
shows *transforms* (*tm-cr j*) *tps t tps'*
unfolding *tm-cr-def* **by** (*tform tps*: *assms*)

2.4.7 Erasing a tape

The next Turing machine overwrites all but the start symbol with blanks. It first performs a carriage return and then writes blanks until it reaches a blank. This only works as intended if there are no gaps, that is, blanks between non-blank symbols.

definition *tm-erase* :: *tapeidx* $\Rightarrow$ *machine* **where**
tm-erase j $\equiv$ *tm-cr j* ;; *tm-const-until j j* $\{\square\}$ $\square$

lemma *tm-erase-tm*: $G \geq 4 \Longrightarrow 0 < j \Longrightarrow j < k \Longrightarrow$ *turing-machine k G* (*tm-erase j*)
unfolding *tm-erase-def* **using** *tm-cr-tm tm-const-until-tm* **by** *simp*

lemma *transforms-tm-eraseI* [*transforms-intros*]:
assumes $j < length\ tps$
and *proper-symbols zs*

```
    and tps ::: j = ⌊zs⌋
    and t = tps :#: j + length zs + 3
    and tps' = tps[j := (⌊[]⌋, Suc (length zs))]
  shows transforms (tm-erase j) tps t tps'
  unfolding tm-erase-def
proof (tform tps: assms time: assms(4))
  show clean-tape (tps ! j)
    using assms contents-clean-tape' by simp
  show rneigh (tps[j := tps ! j |#=| 1] ! j) {□} (length zs)
    using assms contents-clean-tape' by (intro rneighI) auto
  show tps' = tps
    [j := tps ! j |#=| 1, j := tps[j := tps ! j |#=| 1] ! j |+| length zs,
     j := constplant (tps[j := tps ! j |#=| 1] ! j) □ (length zs)]
  proof –
    have (⌊[]⌋, Suc (length zs)) = constplant (⌊zs⌋, Suc 0) □ (length zs)
      using transplant-def contents-def by auto
    then show ?thesis
      using assms by simp
  qed
qed
```

The next TM returns to the leftmost blank symbol after erasing the tape.

```
definition tm-erase-cr :: tapeidx ⇒ machine where
  tm-erase-cr j ≡ tm-erase j ;; tm-cr j

lemma tm-erase-cr-tm:
  assumes G ≥ 4 and 0 < j and j < k
  shows turing-machine k G (tm-erase-cr j)
  using tm-erase-cr-def tm-cr-tm tm-erase-tm assms by simp

lemma transforms-tm-erase-crI [transforms-intros]:
  assumes j < length tps
    and proper-symbols zs
    and tps ::: j = ⌊zs⌋
    and t = tps :#: j + 2 * length zs + 6
    and tps' = tps[j := (⌊[]⌋, 1)]
  shows transforms (tm-erase-cr j) tps t tps'
  unfolding tm-erase-cr-def
  by (tform tps: assms time: assms(4))
```

2.4.8 Writing a symbol sequence

The Turing machine in this section writes a hard-coded symbol sequence to a tape. It is like *tm-write-repeat* except with an arbitrary symbol sequence.

```
fun tm-print :: tapeidx ⇒ symbol list ⇒ machine where
  tm-print j [] = [] |
  tm-print j (z # zs) = tm-char j z ;; tm-print j zs

lemma tm-print-tm:
  assumes 0 < j and j < k and G ≥ 4 and ∀ i<length zs. zs ! i < G
  shows turing-machine k G (tm-print j zs)
  using assms(4)
proof (induction zs)
  case Nil
  then show ?case
    using assms by auto
next
  case (Cons z zs)
  then have turing-machine k G (tm-char j z)
    using assms tm-char-tm by auto
  then show ?case
    using assms Cons by fastforce
qed
```

The result of writing the symbols *zs* to a tape *tp*:

```
definition inscribe :: tape ⇒ symbol list ⇒ tape where
  inscribe tp zs ≡
    (λi. if snd tp ≤ i ∧ i < snd tp + length zs then zs ! (i − snd tp) else fst tp i,
     snd tp + length zs)

lemma inscribe-Nil: inscribe tp [] = tp
proof −
  have (λi. if snd tp ≤ i ∧ i < snd tp then [] ! (i − snd tp) else fst tp i) = fst tp
    by auto
  then show ?thesis
    unfolding inscribe-def by simp
qed

lemma inscribe-Cons: inscribe ((fst tp)(snd tp := z), Suc (snd tp)) zs = inscribe tp (z # zs)
  using inscribe-def by auto

lemma inscribe-contents: inscribe (⌊ys⌋, Suc (length ys)) zs = (⌊ys @ zs⌋, Suc (length ys + length zs))
  (is ?lhs = ?rhs)
proof
  show snd ?lhs = snd ?rhs
    using inscribe-def contents-def by simp
  show fst ?lhs = fst ?rhs
  proof
    fix i :: nat
    consider
        i = 0
      | 0 < i ∧ i < Suc (length ys)
      | Suc (length ys) ≤ i ∧ i < Suc (length ys + length zs)
      | Suc (length ys + length zs) ≤ i
      by linarith
    then show fst ?lhs i = fst ?rhs i
    proof (cases)
      case 1
      then show ?thesis
        using inscribe-def contents-def by simp
    next
      case 2
      then have fst ?lhs i = ⌊ys⌋ i
        using inscribe-def by simp
      then have lhs: fst ?lhs i = ys ! (i − 1)
        using 2 contents-def by simp
      have fst ?rhs i = (ys @ zs) ! (i − 1)
        using 2 contents-def by simp
      then have fst ?rhs i = ys ! (i − 1)
        using 2 by (metis Suc-diff-1 not-less-eq nth-append)
      then show ?thesis
        using lhs by simp
    next
      case 3
      then show ?thesis
        using contents-def inscribe-def
        by (smt (verit, del-insts) One-nat-def add.commute diff-Suc-eq-diff-pred fst-conv length-append less-Suc0
          less-Suc-eq-le less-diff-conv2 nat.simps(3) not-less-eq nth-append plus-1-eq-Suc snd-conv)
    next
      case 4
      then show ?thesis
        using contents-def inscribe-def
        by simp
    qed
  qed
qed
```

lemma *inscribe-contents-Nil*: *inscribe* (⌊[]⌋, *Suc 0*) *zs* = (⌊*zs*⌋, *Suc* (*length zs*))
  using *inscribe-def contents-def* **by** *auto*

lemma *transforms-tm-print*:
  assumes *j* < *length tps*
  shows *transforms* (*tm-print j zs*) *tps* (*length zs*) (*tps*[*j* := *inscribe* (*tps* ! *j*) *zs*])
  using *assms*
proof (*induction zs arbitrary*: *tps*)
  case *Nil*
  then show *?case*
    using *inscribe-Nil transforms-Nil* **by** *simp*
next
  case (*Cons z zs*)
  have *transforms* (*tm-char j z* ;; *tm-print j zs*) *tps* (*length* (*z* # *zs*)) (*tps*[*j* := *inscribe* (*tps* ! *j*) (*z* # *zs*)])
  proof (*tform tps*: *Cons*)
    let *?tps* = *tps*[*j* := *tps* ! *j* |:=| *z* |+| *1*]
    have *transforms* (*tm-print j zs*) *?tps* (*length zs*) (*?tps*[*j* := *inscribe* (*?tps* ! *j*) *zs*])
      using *Cons* **by** (*metis length-list-update*)
    moreover have (*?tps*[*j* := *inscribe* (*?tps* ! *j*) *zs*]) = (*tps*[*j* := *inscribe* (*tps* ! *j*) (*z* # *zs*)])
      using *inscribe-Cons Cons.prems* **by** *simp*
    ultimately show *transforms* (*tm-print j zs*) *?tps* (*length zs*) (*tps*[*j* := *inscribe* (*tps* ! *j*) (*z* # *zs*)])
      by *simp*
  qed
  then show *transforms* (*tm-print j* (*z* # *zs*)) *tps* (*length* (*z* # *zs*)) (*tps*[*j* := *inscribe* (*tps* ! *j*) (*z* # *zs*)])
    by *simp*
qed

lemma *transforms-tm-printI* [*transforms-intros*]:
  assumes *j* < *length tps* **and** *tps'* = (*tps*[*j* := *inscribe* (*tps* ! *j*) *zs*])
  shows *transforms* (*tm-print j zs*) *tps* (*length zs*) *tps'*
  using *assms transforms-tm-print* **by** *simp*

2.4.9 Setting the tape contents to a symbol sequence

The following Turing machine erases the tape, then prints a hard-coded symbol sequence, and then performs a carriage return. It thus sets the tape contents to the symbol sequence.

definition *tm-set* :: *tapeidx* ⇒ *symbol list* ⇒ *machine* **where**
  tm-set j zs ≡ *tm-erase-cr j* ;; *tm-print j zs* ;; *tm-cr j*

lemma *tm-set-tm*:
  assumes *0* < *j* **and** *j* < *k* **and** *G* ≥ *4* **and** ∀ *i*<*length zs*. *zs* ! *i* < *G*
  shows *turing-machine k G* (*tm-set j zs*)
  unfolding *tm-set-def* **using** *assms tm-print-tm tm-erase-cr-tm tm-cr-tm* **by** *simp*

lemma *transforms-tm-setI* [*transforms-intros*]:
  assumes *j* < *length tps*
    and *clean-tape* (*tps* ! *j*)
    and *proper-symbols ys*
    and *proper-symbols zs*
    and *tps* ::: *j* = ⌊*ys*⌋
    and *t* = *8* + *tps* :#: *j* + *2* ∗ *length ys* + *Suc* (*2* ∗ *length zs*)
    and *tps'* = *tps*[*j* := (⌊*zs*⌋, *1*)]
  shows *transforms* (*tm-set j zs*) *tps t tps'*
  unfolding *tm-set-def*
proof (*tform tps*: *assms*(*1*−*5*))
  show *clean-tape*
    (*tps*[*j* := (⌊[]⌋, *1*),
        j := *inscribe* (*tps*[*j* := (⌊[]⌋, *1*)] ! *j*) *zs*] ! *j*)
    using *assms inscribe-contents-Nil clean-contents-proper*[*OF assms*(*4*)] **by** *simp*
  show *tps'* = *tps*
     [*j* := (⌊[]⌋, *1*), *j* := *inscribe* (*tps*[*j* := (⌊[]⌋, *1*)] ! *j*) *zs*,
      j := *tps*[*j* := (⌊[]⌋, *1*), *j* := *inscribe* (*tps*[*j* := (⌊[]⌋, *1*)] ! *j*) *zs*] ! *j* |#=| *1*]
    using *assms inscribe-def clean-tape-def assms contents-def inscribe-contents-Nil* **by** *simp*

```
  show t = tps :#: j + 2 * length ys + 6 + length zs +
      (tps[j := (⌊[]⌋, 1),
           j := inscribe (tps[j := (⌊[]⌋, 1)] ! j) zs] :#: j + 2)
    using assms inscribe-def by simp
qed
```

2.4.10 Comparing two tapes

The next Turing machine compares the contents of two tapes j_1 and j_2 and writes to tape j_3 either a **1** or a □ depending on whether the tapes are equal or not. The next command does all the work. It scans both tapes left to right and halts if it encounters a blank on both tapes, which means the tapes are equal, or two different symbols, which means the tapes are unequal. It only works for contents without blanks.

```
definition cmd-cmp :: tapeidx ⇒ tapeidx ⇒ tapeidx ⇒ command where
  cmd-cmp j1 j2 j3 rs ≡
    if rs ! j1 ≠ rs ! j2
    then (1, map (λi. (if i = j3 then □ else rs ! i, Stay)) [0..<length rs])
    else if rs ! j1 = □ ∨ rs ! j2 = □
      then (1, map (λi. (if i = j3 then 1 else rs ! i, Stay)) [0..<length rs])
      else (0, map (λi. (rs ! i, if i = j1 ∨ i = j2 then Right else Stay)) [0..<length rs])

lemma sem-cmd-cmp1:
  assumes length tps = k
    and j1 < k and j2 < k and j3 < k
    and tps :.: j1 ≠ tps :.: j2
  shows sem (cmd-cmp j1 j2 j3) (0, tps) = (1, tps[j3 := tps ! j3 |:=| □])
  unfolding cmd-cmp-def using assms tapes-at-read' act-Stay read-length by (intro semI) auto

lemma sem-cmd-cmp2:
  assumes length tps = k
    and j1 < k and j2 < k and j3 < k
    and tps :.: j1 = tps :.: j2 and tps :.: j1 = □ ∨ tps :.: j2 = □
  shows sem (cmd-cmp j1 j2 j3) (0, tps) = (1, tps[j3 := tps ! j3 |:=| 1])
  unfolding cmd-cmp-def using assms tapes-at-read' act-Stay read-length by (intro semI) auto

lemma sem-cmd-cmp3:
  assumes length tps = k
    and j2 ≠ j3 and j1 ≠ j3 and j1 < k and j2 < k and j3 < k
    and tps :.: j1 = tps :.: j2 and tps :.: j1 ≠ □ ∧ tps :.: j2 ≠ □
  shows sem (cmd-cmp j1 j2 j3) (0, tps) = (0, tps[j1 := tps ! j1 |+| 1, j2 := tps ! j2 |+| 1])
proof (rule semI)
  show proper-command k (cmd-cmp j1 j2 j3)
    using cmd-cmp-def by simp
  show length tps = k
    using assms(1) .
  show length (tps[j1 := tps ! j1 |+| 1, j2 := tps ! j2 |+| 1]) = k
    using assms(1) by simp
  show fst ((cmd-cmp j1 j2 j3) (read tps)) = 0
    unfolding cmd-cmp-def using assms tapes-at-read' by simp
  show act (cmd-cmp j1 j2 j3 (read tps) [!] j) (tps ! j) = tps[j1 := tps ! j1 |+| 1, j2 := tps ! j2 |+| 1] ! j
      if j < k for j
    unfolding cmd-cmp-def
    using assms tapes-at-read' that act-Stay act-Right read-length
    by (cases j1 = j2) simp-all
qed

definition tm-cmp :: tapeidx ⇒ tapeidx ⇒ tapeidx ⇒ machine where
  tm-cmp j1 j2 j3 ≡ [cmd-cmp j1 j2 j3]

lemma tm-cmp-tm:
  assumes k ≥ 2 and j3 > 0 and G ≥ 4
  shows turing-machine k G (tm-cmp j1 j2 j3)
  unfolding tm-cmp-def cmd-cmp-def using assms turing-machine-def by auto
```

lemma *exe-cmd-cmp1*:
  assumes *length tps* = *k*
    and *j1* < *k* **and** *j2* < *k* **and** *j3* < *k*
    and *tps* :.: *j1* ≠ *tps* :.: *j2*
  shows *exe* (*tm-cmp j1 j2 j3*) (*0*, *tps*) = (*1*, *tps*[*j3* := *tps* ! *j3* |:=| □])
  using *tm-cmp-def assms exe-lt-length sem-cmd-cmp1* **by** *simp*

lemma *exe-cmd-cmp2*:
  assumes *length tps* = *k*
    and *j1* < *k* **and** *j2* < *k* **and** *j3* < *k*
    and *tps* :.: *j1* = *tps* :.: *j2* **and** *tps* :.: *j1* = □ ∨ *tps* :.: *j2* = □
  shows *exe* (*tm-cmp j1 j2 j3*) (*0*, *tps*) = (*1*, *tps*[*j3* := *tps* ! *j3* |:=| **1**])
  using *tm-cmp-def assms exe-lt-length sem-cmd-cmp2* **by** *simp*

lemma *exe-cmd-cmp3*:
  assumes *length tps* = *k*
    and *j2* ≠ *j3* **and** *j1* ≠ *j3* **and** *j1* < *k* **and** *j2* < *k* **and** *j3* < *k*
    and *tps* :.: *j1* = *tps* :.: *j2* **and** *tps* :.: *j1* ≠ □ ∧ *tps* :.: *j2* ≠ □
  shows *exe* (*tm-cmp j1 j2 j3*) (*0*, *tps*) = (*0*, *tps*[*j1* := *tps* ! *j1* |+| *1*, *j2* := *tps* ! *j2* |+| *1*])
  using *tm-cmp-def assms exe-lt-length sem-cmd-cmp3* **by** *simp*

lemma *execute-tm-cmp-eq*:
  fixes *tps* :: *tape list*
  assumes *length tps* = *k*
    and *j2* ≠ *j3* **and** *j1* ≠ *j3* **and** *j1* < *k* **and** *j2* < *k* **and** *j3* < *k*
    and *proper-symbols xs*
    and *tps* ! *j1* = (⌊*xs*⌋, *1*)
    and *tps* ! *j2* = (⌊*xs*⌋, *1*)
  shows *execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*) (*Suc* (*length xs*)) =
      (*1*, *tps*[*j1* := *tps* ! *j1* |+| *length xs*, *j2* := *tps* ! *j2* |+| *length xs*, *j3* := *tps* ! *j3* |:=| **1**])
proof −
  have *execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*) *t* = (*0*, *tps*[*j1* := *tps* ! *j1* |+| *t*, *j2* := *tps* ! *j2* |+| *t*])
      if *t* ≤ *length xs* **for** *t*
    using *that*
  proof (*induction t*)
    case *0*
    then show *?case*
      by *simp*
  next
    case (*Suc t*)
    then have *t-less*: *t* < *length xs*
      by *simp*
    have *execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*) (*Suc t*) = *exe* (*tm-cmp j1 j2 j3*) (*execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*)
t)
      by *simp*
    also have ... = *exe* (*tm-cmp j1 j2 j3*) (*0*, *tps*[*j1* := *tps* ! *j1* |+| *t*, *j2* := *tps* ! *j2* |+| *t*])
        (**is** - = *exe* - (*0*, *?tps*))
      using *Suc* **by** *simp*
    also have ... = (*0*, *?tps*[*j1* := *?tps* ! *j1* |+| *1*, *j2* := *?tps* ! *j2* |+| *1*])
    proof −
      have *1*: *?tps* :.: *j1* = *xs* ! *t*
        using *assms*(*1*,*2*,*4*,*8*) *t-less Suc.prems contents-inbounds*
        by (*metis* (*no-types*, *lifting*) *diff-Suc-1 fst-conv length-list-update nth-list-update-eq*
          nth-list-update-neq plus-1-eq-Suc snd-conv zero-less-Suc)
      moreover have *2*: *?tps* :.: *j2* = *xs* ! *t*
        using *t-less assms*(*1*,*5*,*9*) **by** *simp*
      ultimately have *?tps* :.: *j1* = *?tps* :.: *j2*
        by *simp*
      moreover have *?tps* :.: *j1* ≠ *0* ∧ *?tps* :.: *j2* ≠ *0*
        using *1 2 assms*(*7*) *t-less* **by** *auto*
      moreover have *length ?tps* = *k*
        using *assms*(*1*) **by** *simp*
      ultimately show *?thesis*

```
      using assms exe-cmd-cmp3 by blast
    qed
    also have ... = (0, tps[j1 := tps ! j1 |+| Suc t, j2 := tps ! j2 |+| Suc t])
      using assms
      by (smt (verit) Suc-eq-plus1 add.commute fst-conv list-update-overwrite list-update-swap
        nth-list-update-eq nth-list-update-neq snd-conv)
    finally show ?case
      by simp
  qed
  then have execute (tm-cmp j1 j2 j3) (0, tps) (length xs) =
      (0, tps[j1 := tps ! j1 |+| length xs, j2 := tps ! j2 |+| length xs])
    by simp
  then have execute (tm-cmp j1 j2 j3) (0, tps) (Suc (length xs)) =
      exe (tm-cmp j1 j2 j3) (0, tps[j1 := tps ! j1 |+| length xs, j2 := tps ! j2 |+| length xs])
      (is - = exe - (0, ?tps))
    by simp
  also have ... = (1, ?tps[j3 := ?tps ! j3 |:=| 1])
  proof -
    have 1: ?tps :.: j1 = 0
      using assms(1,4,8) contents-outofbounds
      by (metis fst-conv length-list-update lessI nth-list-update-eq nth-list-update-neq plus-1-eq-Suc snd-conv)
    moreover have 2: ?tps :.: j2 = 0
      using assms(1,5,9) by simp
    ultimately have ?tps :.: j1 = ?tps :.: j2 ?tps :.: j1 = □ ∨ ?tps :.: j2 = □
      by simp-all
    moreover have length ?tps = k
      using assms(1) by simp
    ultimately show ?thesis
      using assms exe-cmd-cmp2 by blast
  qed
  also have ... = (1, tps[j1 := tps ! j1 |+| length xs, j2 := tps ! j2 |+| length xs, j3 := tps ! j3 |:=| 1])
    using assms by simp
  finally show ?thesis .
qed

lemma ex-contents-neq:
  assumes proper-symbols xs and proper-symbols ys and xs ≠ ys
  shows ∃ m. m ≤ Suc (min (length xs) (length ys)) ∧ ⌊xs⌋ m ≠ ⌊ys⌋ m
proof -
  consider length xs < length ys | length xs = length ys | length xs > length ys
    by linarith
  then show ?thesis
  proof (cases)
    case 1
    let ?m = length xs
    have ⌊xs⌋ (Suc ?m) = □
      by simp
    moreover have ⌊ys⌋ (Suc ?m) ≠ □
      using 1 assms(2) by (simp add: proper-symbols-ne0)
    ultimately show ?thesis
      using 1 by auto
  next
    case 2
    then have ∃ i<length xs. xs ! i ≠ ys ! i
      using assms by (meson list-eq-iff-nth-eq)
    then show ?thesis
      using contents-def 2 by auto
  next
    case 3
    let ?m = length ys
    have ⌊ys⌋ (Suc ?m) = □
      by simp
    moreover have ⌊xs⌋ (Suc ?m) ≠ □
```

```
      using 3 assms(1) by (simp add: proper-symbols-ne0)
    ultimately show ?thesis
      using 3 by auto
  qed
qed

lemma execute-tm-cmp-neq:
  fixes tps :: tape list
  assumes length tps = k
    and j1 ≠ j2 and j2 ≠ j3 and j1 ≠ j3 and j1 < k and j2 < k and j3 < k
    and proper-symbols xs
    and proper-symbols ys
    and xs ≠ ys
    and tps ! j1 = (⌊xs⌋, 1)
    and tps ! j2 = (⌊ys⌋, 1)
    and m = (LEAST m. m ≤ Suc (min (length xs) (length ys)) ∧ ⌊xs⌋ m ≠ ⌊ys⌋ m)
  shows execute (tm-cmp j1 j2 j3) (0, tps) m =
    (1, tps[j1 := tps ! j1 |+| (m − 1), j2 := tps ! j2 |+| (m − 1), j3 := tps ! j3 |:=| □])
proof −
  have neq: ⌊xs⌋ m ≠ ⌊ys⌋ m
    using ex-contents-neq[OF assms(8−10)] assms(13) by (metis (mono-tags, lifting) LeastI-ex)
  have eq: ⌊xs⌋ i = ⌊ys⌋ i if i < m for i
      using ex-contents-neq[OF assms(8−10)] assms(13) not-less-Least that by (smt (verit) Least-le le-trans
less-imp-le-nat)
  have m > 0
    using neq contents-def gr0I by metis

  have execute (tm-cmp j1 j2 j3) (0, tps) t = (0, tps[j1 := tps ! j1 |+| t, j2 := tps ! j2 |+| t])
      if t < m for t
    using that
  proof (induction t)
    case 0
    then show ?case
      by simp
  next
    case (Suc t)
    have execute (tm-cmp j1 j2 j3) (0, tps) (Suc t) = exe (tm-cmp j1 j2 j3) (execute (tm-cmp j1 j2 j3) (0, tps)
t)
      by simp
    also have ... = exe (tm-cmp j1 j2 j3) (0, tps[j1 := tps ! j1 |+| t, j2 := tps ! j2 |+| t])
        (is - = exe - (0, ?tps))
      using Suc by simp
    also have ... = (0, ?tps[j1 := ?tps ! j1 |+| 1, j2 := ?tps ! j2 |+| 1])
    proof −
      have 1: ?tps :.: j1 = ⌊xs⌋ (Suc t)
        using assms(1,2,5,11) by simp
      moreover have 2: ?tps :.: j2 = ⌊ys⌋ (Suc t)
        using assms(1,6,12) by simp
      ultimately have ?tps :.: j1 = ?tps :.: j2
        using Suc eq by simp
      moreover from this have ?tps :.: j1 ≠ □ ∧ ?tps :.: j2 ≠ □
        using 1 2 assms neq Suc.prems contents-def
        by (smt (verit) Suc-leI Suc-le-lessD Suc-lessD diff-Suc-1 le-trans less-nat-zero-code zero-less-Suc)
      moreover have length ?tps = k
        using assms(1) by simp
      ultimately show ?thesis
        using assms exe-cmd-cmp3 by blast
    qed
    also have ... = (0, tps[j1 := tps ! j1 |+| Suc t, j2 := tps ! j2 |+| Suc t])
      using assms
      by (smt (verit) Suc-eq-plus1 add.commute fst-conv list-update-overwrite list-update-swap
        nth-list-update-eq nth-list-update-neq snd-conv)
    finally show ?case
```

by *simp*
qed
then have *execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*) (*m* − *1*) =
(*0*, *tps*[*j1* := *tps* ! *j1* |+| (*m* − *1*), *j2* := *tps* ! *j2* |+| (*m* − *1*)])
using ‹*m* > *0*› **by** *simp*
then have *execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*) *m* =
exe (*tm-cmp j1 j2 j3*) (*0*, *tps*[*j1* := *tps* ! *j1* |+| (*m* − *1*), *j2* := *tps* ! *j2* |+| (*m* − *1*)])
using ‹*m* > *0*› **by** (*metis contents-at-0 diff-Suc-1 execute.elims neq*)
then show *execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*) *m* =
(*1*, *tps*[*j1* := *tps* ! *j1* |+| (*m* − *1*), *j2* := *tps* ! *j2* |+| (*m* − *1*), *j3* := *tps* ! *j3* |:=| □])
using *exe-cmd-cmp1 assms* ‹*0* < *m*›
by (*smt* (*verit*) *One-nat-def Suc-diff-Suc diff-zero fst-conv length-list-update neq nth-list-update-eq nth-list-update-neq plus-1-eq-Suc snd-conv*)
qed

lemma *transforms-tm-cmpI* [*transforms-intros*]:
fixes *tps* :: *tape list*
assumes *length tps* = *k*
and *j1* ≠ *j2* **and** *j2* ≠ *j3* **and** *j1* ≠ *j3* **and** *j1* < *k* **and** *j2* < *k* **and** *j3* < *k*
and *proper-symbols xs*
and *proper-symbols ys*
and *tps* ! *j1* = (⌊*xs*⌋, *1*)
and *tps* ! *j2* = (⌊*ys*⌋, *1*)
and *t* = *Suc* (*min* (*length xs*) (*length ys*))
and *b* = (*if xs* = *ys then* **1** *else* □)
and *m* =
(*if xs* = *ys*
then Suc (*length xs*)
else (*LEAST m*. *m* ≤ *Suc* (*min* (*length xs*) (*length ys*)) ∧ ⌊*xs*⌋ *m* ≠ ⌊*ys*⌋ *m*))
and *tps′* = *tps*[*j1* := (⌊*xs*⌋, *m*), *j2* := (⌊*ys*⌋, *m*), *j3* := *tps* ! *j3* |:=| *b*]
shows *transforms* (*tm-cmp j1 j2 j3*) *tps t tps′*
proof (*cases xs* = *ys*)
case *True*
then have *m*: *m* = *Suc* (*length xs*)
using *assms*(*14*) **by** *simp*
have *execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*) (*Suc* (*length xs*)) =
(*1*, *tps*[*j1* := *tps* ! *j1* |+| *length xs*, *j2* := *tps* ! *j2* |+| *length xs*, *j3* := *tps* ! *j3* |:=| **1**])
using *execute-tm-cmp-eq assms True* **by** *blast*
then have *execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*) *m* =
(*1*, *tps*[*j1* := *tps* ! *j1* |+| (*m* − *1*), *j2* := *tps* ! *j2* |+| (*m* − *1*), *j3* := *tps* ! *j3* |:=| *b*])
using *m assms*(*13*) *True diff-Suc-1* **by** *simp*
moreover have *m* ≤ *t*
using *True assms*(*12*) *m* **by** *simp*
ultimately show *?thesis*
using *transitsI tm-cmp-def transforms-def assms True*
by (*metis* (*no-types*, *lifting*) *One-nat-def add.commute diff-Suc-1 fst-conv list.size*(*3*) *list.size*(*4*) *plus-1-eq-Suc snd-conv*)
next
case *False*
then have *m*: *m* = (*LEAST m*. *m* ≤ *Suc* (*min* (*length xs*) (*length ys*)) ∧ ⌊*xs*⌋ *m* ≠ ⌊*ys*⌋ *m*)
using *assms*(*14*) **by** *simp*
then have *execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*) *m* =
(*1*, *tps*[*j1* := *tps* ! *j1* |+| (*m* − *1*), *j2* := *tps* ! *j2* |+| (*m* − *1*), *j3* := *tps* ! *j3* |:=| □])
using *False assms execute-tm-cmp-neq* **by** *blast*
then have *execute* (*tm-cmp j1 j2 j3*) (*0*, *tps*) *m* =
(*1*, *tps*[*j1* := *tps* ! *j1* |+| (*m* − *1*), *j2* := *tps* ! *j2* |+| (*m* − *1*), *j3* := *tps* ! *j3* |:=| *b*])
using *False* **by** (*simp add*: *assms*(*13*))
moreover have *m* ≤ *t*
using *ex-contents-neq*[*OF assms*(*8*,*9*)] *False assms*(*12*) *m* **by** (*metis* (*mono-tags*, *lifting*) *LeastI*)
ultimately show *?thesis*
using *transitsI tm-cmp-def transforms-def assms False*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-eq-plus1 Suc-pred add.commute execute.simps*(*1*) *fst-eqD list.size*(*3*) *list.size*(*4*) *not-gr0 numeral-One snd-conv zero-neq-numeral*)

qed

The next Turing machine extends *tm-cmp* by a carriage return on tapes j_1 and j_2, ensuring that the next command finds the tape heads in a well-specified position. This makes the TM easier to reuse.

definition *tm-equals* :: *tapeidx* ⇒ *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-equals j1 j2 j3 ≡ *tm-cmp j1 j2 j3* ;; *tm-cr j1* ;; *tm-cr j2*

lemma *tm-equals-tm*:
assumes $k \geq 2$ **and** $j3 > 0$ **and** $G \geq 4$
shows *turing-machine k G* (*tm-equals j1 j2 j3*)
unfolding *tm-equals-def* **using** *tm-cmp-tm tm-cr-tm assms* **by** *simp*

We analyze the behavior of *tm-equals* inside a locale. This is how we will typically proceed for Turing machines that are composed of more than two TMs. The locale is parameterized by the TM's parameters, which in the present case means the three tape indices j_1, j_2, and j_3. Inside the locale the TM is decomposed such that proofs of *transforms* only involve two TMs combined by one of the three control structures (sequence, branch, loop). In the current example we have three TMs named *tm1*, *tm2*, *tm3*, where *tm3* is just *tm-equals*. Furthermore there will be lemmas *tm1*, *tm2*, *tm3* describing, in terms of *transforms*, the behavior of the respective TMs. For this we define three tape lists *tps1*, *tps2*, *tps3*.

This naming scheme creates many name clashes for things that only have a single use. That is the reason for the encapsulation in a locale.

Afterwards this locale is interpreted, just once in lemma *transforms-tm-equalsI*, to prove the semantics and running time of *tm-equals*.

locale *turing-machine-equals* =
fixes *j1 j2 j3* :: *tapeidx*
begin

definition *tm1* ≡ *tm-cmp j1 j2 j3*
definition *tm2* ≡ *tm1* ;; *tm-cr j1*
definition *tm3* ≡ *tm2* ;; *tm-cr j2*

lemma *tm3-eq-tm-equals*: *tm3* = *tm-equals j1 j2 j3*
unfolding *tm3-def tm2-def tm1-def tm-equals-def* **by** *simp*

context
fixes *tps0* :: *tape list* **and** *k t b* :: *nat* **and** *xs ys* :: *symbol list*
assumes *jk* [*simp*]: *length tps0* = *k j1* ≠ *j2 j2* ≠ *j3 j1* ≠ *j3 j1* < *k j2* < *k j3* < *k*
and *proper*: *proper-symbols xs proper-symbols ys*
and *t*: *t* = *Suc* (*min* (*length xs*) (*length ys*))
and *b*: *b* = (*if xs* = *ys then 3 else 0*)
assumes *tps0*:
tps0 ! *j1* = (⌊*xs*⌋, *1*)
tps0 ! *j2* = (⌊*ys*⌋, *1*)
begin

definition *m* ≡
(*if xs* = *ys*
then Suc (*length xs*)
else (*LEAST m*. *m* ≤ *Suc* (*min* (*length xs*) (*length ys*)) ∧ ⌊*xs*⌋ *m* ≠ ⌊*ys*⌋ *m*))

lemma *m-gr-0*: *m* > *0*
proof −
have ⌊*xs*⌋ *m* ≠ ⌊*ys*⌋ *m* **if** *xs* ≠ *ys*
using *ex-contents-neq LeastI-ex m-def proper that* **by** (*metis* (*mono-tags*, *lifting*))
then show *m* > *0*
using *m-def* **by** (*metis contents-at-0 gr0I less-Suc-eq-0-disj*)
qed

lemma *m-le-t*: *m* ≤ *t*
proof (*cases xs* = *ys*)
case *True*

then show *?thesis*
  using *t m-def* **by** *simp*
next
  case *False*
  then have $m \leq Suc\ (min\ (length\ xs)\ (length\ ys))$
    using *ex-contents-neq False proper m-def* **by** (*metis* (*mono-tags*, *lifting*) *LeastI-ex*)
  then show *?thesis*
    using *t* **by** *simp*
qed

definition $tps1 \equiv tps0[j1 := (\lfloor xs \rfloor, m), j2 := (\lfloor ys \rfloor, m), j3 := tps0\ !\ j3\ |{:=}|\ b]$

lemma *tm1* [*transforms-intros*]: *transforms tm1 tps0 t tps1*
  unfolding *tm1-def*
proof (*tform tps*: *tps0 tps1-def m-def b time*: *t*)
  show *proper-symbols xs proper-symbols ys*
    using *proper* **by** *simp-all*
qed

definition $tps2 \equiv tps0[j1 := (\lfloor xs \rfloor, 1), j2 := (\lfloor ys \rfloor, m), j3 := tps0\ !\ j3\ |{:=}|\ b]$

lemma *tm2*:
  assumes $ttt = t + m + 2$
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def*
proof (*tform tps*: *tps1-def*)
  show *clean-tape* (*tps1* ! *j1*)
    using *tps1-def clean-contents-proper jk proper*(*1*)
    by (*metis nth-list-update-eq nth-list-update-neq*)
  show $ttt = t + (tps1\ {:}\#{:}\ j1 + 2)$
    using *tps1-def tps0 jk assms*
    by (*metis* (*no-types*, *lifting*) *ab-semigroup-add-class.add-ac*(*1*) *nth-list-update-eq nth-list-update-neq snd-conv*)
  show $tps2 = tps1[j1 := tps1\ !\ j1\ |\#{=}|\ 1]$
    unfolding *tps2-def tps1-def* **by** (*simp add*: *list-update-swap*[*of j1*])
qed

lemma *tm2'* [*transforms-intros*]: *transforms tm2 tps0* $(2 * t + 2)$ *tps2*
  using *m-le-t tm2 transforms-monotone* **by** *simp*

definition $tps3 \equiv tps0[j1 := (\lfloor xs \rfloor, 1), j2 := (\lfloor ys \rfloor, 1), j3 := tps0\ !\ j3\ |{:=}|\ b]$

lemma *tm3*:
  assumes $ttt = 2 * t + m + 4$
  shows *transforms tm3 tps0 ttt tps3*
  unfolding *tm3-def*
proof (*tform tps*: *tps2-def tps3-def*)
  have $*$: $tps2\ !\ j2 = (\lfloor ys \rfloor, m)$
    using *tps2-def* **by** (*simp add*: *nth-list-update-neq'*)
  then show *clean-tape* (*tps2* ! *j2*)
    using *clean-contents-proper proper*(*2*) **by** *simp*
  show $ttt = 2 * t + 2 + (tps2\ {:}\#{:}\ j2 + 2)$
    using *assms* $*$ **by** *simp*
  show $tps3 = tps2[j2 := tps2\ !\ j2\ |\#{=}|\ 1]$
    unfolding *tps3-def tps2-def* **by** (*simp add*: *list-update-swap*[*of j2*])
qed

definition $tps3' \equiv tps0[j3 := tps0\ !\ j3\ |{:=}|\ b]$

lemma *tm3'*: *transforms tm3 tps0* $(3 * min\ (length\ xs)\ (length\ ys) + 7)$ *tps3'*
proof −
  have $tps3' = tps3$
    using *tps3'-def tps3-def tps0 jk* **by** (*metis list-update-id*)
  then show *?thesis*

using *m-le-t tm3 transforms-monotone t* **by** *simp*
qed

end

end

lemma *transforms-tm-equalsI* [*transforms-intros*]:
fixes *j1 j2 j3* :: *tapeidx*
fixes *tps tps'* :: *tape list* **and** *k* :: *nat* **and** *xs ys* :: *symbol list* **and** *b* :: *symbol*
assumes *length tps* = *k j1* ≠ *j2 j2* ≠ *j3 j1* ≠ *j3 j1* < *k j2* < *k j3* < *k*
and *proper-symbols xs proper-symbols ys*
and *b* = (*if xs* = *ys then* **1** *else* □)
assumes
tps ! *j1* = (⌊*xs*⌋, *1*)
tps ! *j2* = (⌊*ys*⌋, *1*)
assumes *ttt* = *3* ∗ *min* (*length xs*) (*length ys*) + *7*
assumes *tps'* = *tps*
[*j3* := *tps* ! *j3* |:=| *b*]
shows *transforms* (*tm-equals j1 j2 j3*) *tps ttt tps'*
proof −
interpret *loc*: *turing-machine-equals j1 j2 j3* **.**
show *?thesis*
using *assms loc.tm3' loc.tm3-eq-tm-equals loc.tps3'-def* **by** *simp*
qed

2.4.11 Computing the identity function

In order to compute the identity function, a Turing machine can just copy the input tape to the output tape:

definition *tm-id* :: *machine* **where**
tm-id ≡ *tm-cp-until 0 1* {□}

lemma *tm-id-tm*:
assumes *1* < *k* **and** *G* ≥ *4*
shows *turing-machine k G tm-id*
unfolding *tm-id-def* **using** *assms tm-cp-until-tm* **by** *simp*

lemma *transforms-tm-idI*:
fixes *zs* :: *symbol list* **and** *k* :: *nat* **and** *tps* :: *tape list*
assumes *1* < *k*
and *proper-symbols zs*
and *tps* = *snd* (*start-config k zs*)
and *tps'* = *tps*[*0* := (⌊*zs*⌋, (*Suc* (*length zs*))), *1* := (⌊*zs*⌋, (*Suc* (*length zs*)))]
shows *transforms tm-id tps* (*Suc* (*Suc* (*length zs*))) *tps'*
proof −
let *?n* = *Suc* (*length zs*)
define *tps2* **where**
tps2 = *tps*[*0* := *tps* ! *0* |+| (*Suc* (*length zs*)), *1* := *implant* (*tps* ! *0*) (*tps* ! *1*) (*Suc* (*length zs*))]
have *1*: *rneigh* (*tps* ! *0*) {□} *?n*
proof (*rule rneighI*)
show (*tps* ::: *0*) (*tps* :#: *0* + *Suc* (*length zs*)) ∈ {□}
using *start-config2 start-config3 assms* **by** (*simp add*: *start-config-def*)
show ⋀*n'*. *n'* < *Suc* (*length zs*) ⟹ (*tps* ::: *0*) (*tps* :#: *0* + *n'*) ∉ {□}
using *start-config2 start-config3 start-config-pos assms*
by (*metis One-nat-def Suc-lessD add-cancel-right-left diff-Suc-1 less-Suc-eq-0-disj less-Suc-eq-le not-one-less-zero singletonD*)
qed
have *2*: *length tps* = *k*
using *assms*(*1*,*3*) **by** (*simp add*: *start-config-length*)
have ∗∗: *transforms tm-id tps* (*Suc ?n*) *tps2*
unfolding *tm-id-def* **using** *transforms-tm-cp-untilI*[*OF - assms*(*1*) *2 1 - tps2-def*] *assms*(*1*) **by** *simp*

```
  have 0: tps ! 0 = (⌊zs⌋, 0)
    using assms start-config-def contents-def by auto
  moreover have tps ! 1 = (⌊[]⌋, 0)
    using assms start-config-def contents-def by auto
  moreover have implant (⌊zs⌋, 0) (⌊[]⌋, 0) ?n = (⌊zs⌋, ?n)
    by (rule implantI''') simp-all
  ultimately have implant (tps ! 0) (tps ! 1) ?n = (⌊zs⌋, ?n)
    by simp
  then have tps2 = tps[0 := tps ! 0 |+| ?n, 1 := (⌊zs⌋, ?n)]
    using tps2-def by simp
  then have tps2 = tps[0 := (⌊zs⌋, ?n), 1 := (⌊zs⌋, ?n)]
    using 0 by simp
  then have tps2 = tps'
    using assms(4) by simp
  then show ?thesis
    using ** by simp
qed
```

The identity function is computable with a time bound of $n + 2$.

```
lemma computes-id: computes-in-time 2 tm-id id (λn. Suc (Suc n))
proof
  fix x :: string
  let ?zs = string-to-symbols x
  let ?start = snd (start-config 2 ?zs)
  let ?T = λn. Suc (Suc n)
  let ?tps = ?start[0 := (⌊?zs⌋, (Suc (length ?zs))), 1 := (⌊?zs⌋, (Suc (length ?zs)))]
  have proper-symbols ?zs
    by simp
  then have transforms tm-id ?start (Suc (Suc (length ?zs))) ?tps
    using transforms-tm-idI One-nat-def less-2-cases-iff by blast
  then have transforms tm-id ?start (?T (length x)) ?tps
    by simp
  moreover have ?tps ::: 1 = string-to-contents (id x)
    by (auto simp add: start-config-length)
  ultimately show ∃ tps. tps ::: 1 = string-to-contents (id x) ∧ transforms tm-id ?start (?T (length x)) tps
    by auto
qed

end
```

2.5 Memorizing in states

```
theory Memorizing
  imports Elementary
begin
```

Some Turing machines are best described by allowing them to memorize values in their states. For example, a TM that adds two binary numbers could memorize the carry bit in states. In the textbook definition of TMs, with arbitrary state space, this can be represented by a state space of the form $Q \times \{0, 1\}$, where 0 and 1 represent the memorized values. In our simplified definition of TMs, where the state space is an interval of natural numbers, this does not work. However, there is a workaround. Since we can have arbitrarily many tapes, we can make the TM store this value on an additional tape. Such a memorization tape could be used to write/read a symbol representing the memorized value. The tape head would never move on such a tape. The behavior of the TM can then depend on the memorized value.

By adding several such tapes we can even have more than one value stored simultaneously as well. However, this method increases the number of tapes, and one part of the proof of the Cook-Levin theorem is showing that every TM can be simulated on a two-tape TM (see Chapter 5.3). How to remove such memorization tapes again without changing the behavior of the TM is the subject of this section.

The straightforward idea is to multiply the states by the number of possible values. So if the original TM has Q non-halting states and memorizes G different values, the new TM has $Q \cdot G$ non-halting states. It would be natural to map a pair (q, g) of state and memorized value to $q \cdot G + g$ or to $g \cdot Q + q$. However, there is a small technical problem.

The memorization tape is initialized, like all tapes in a start configuration, with the head on the leftmost cell, which contains the start symbol. Thus the initially memorized value is the number 1 representing $\triangleright$. The new TM must start in the start state, which we have fixed at 0. Thus the state-value pair $(0, 1)$ must be mapped to 0, which neither of the two natural mappings does. Our workaround is to use the mapping $(q, g) \mapsto ((g - 1) \bmod G) \cdot Q + q$.

The following function maps a Turing machine M that memorizes one value from $\{0, \ldots, G - 1\}$ on its last tape to a TM that has one tape less, has G times the number of non-halting states, and behaves just like M. The name "cartesian" for this function is just a funny term I made up.

```
definition cartesian :: machine ⇒ nat ⇒ machine where
  cartesian M G ≡
   concat
    (map (λh. map (λq rs.
                    let (q', as) = (M ! q) (rs @ [(h + 1) mod G])
                    in (if q' = length M then G ∗ length M else (fst (last as) + G − 1) mod G ∗ length M + q',
                        butlast as))
              [0..<length M])
      [0..<G])

lemma length-concat-const:
  assumes ⋀h. length (f h) = c
  shows length (concat (map f [0..<G])) = G ∗ c
  using assms by (induction G; simp)

lemma length-cartesian: length (cartesian M G) = G ∗ length M
  using cartesian-def by (simp add: length-concat-const)

lemma concat-nth:
  assumes ⋀h. length (f h) = c
    and xs = concat (map f [0..<G])
    and h < G
    and q < c
  shows xs ! (h ∗ c + q) = f h ! q
  using assms(2,3)
proof (induction G arbitrary: xs)
  case 0
  then show ?case
    by simp
next
  case (Suc G)
  then have 1: xs = concat (map f [0..<G]) @ f G (is xs = ?ys @ f G)
    by simp
  show ?case
  proof (cases h < G)
    case True
    then have h ∗ c ≤ (G − 1) ∗ c
      by auto
    then have h ∗ c ≤ G ∗ c − c
      using True by (simp add: diff-mult-distrib)
    then have h ∗ c + q ≤ G ∗ c − c + q
      using assms(4) by simp
    then have h ∗ c + q < G ∗ c
      using assms(4) True ‹h ∗ c ≤ (G − 1) ∗ c› mult-eq-if by force
    then have h ∗ c + q < length ?ys
      using length-concat-const assms by metis
    then have xs ! (h ∗ c + q) = ?ys ! (h ∗ c + q)
      using 1 by (simp add: nth-append)
    then show ?thesis
      using Suc True by simp
  next
    case False
```

```
    then show ?thesis
      using 1 Suc.prems(2) assms(1)
      by (metis add-diff-cancel-left' length-concat-const less-SucE not-add-less1 nth-append)
  qed
qed

lemma cartesian-at:
  assumes M' = cartesian M b and h < b and q < length M
  shows (M' ! (h * length M + q)) rs =
    (let (q', as) = (M ! q) (rs @ [(h + 1) mod b])
     in (if q' = length M then b * length M else (fst (last as) + b − 1) mod b * length M + q',
         butlast as))
proof −
  define f where f =
   (λh. map (λq. λrs.
      let (q', as) = (M ! q) (rs @ [(h + 1) mod b])
      in (if q' = length M then b * length M else (fst (last as) + b − 1) mod b * length M + q',
          butlast as))
    [0..<length M])
  then have length (f h) = length M for h
    by simp
  moreover have M' = concat (map f [0..<b])
    using assms(1) cartesian-def f-def by simp
  ultimately have M' ! (h * length M + q) = f h ! q
    using concat-nth assms(2,3) by blast
  then show ?thesis
    using f-def by (simp add: assms(3))
qed

lemma concat-nth-ex:
  assumes ⋀h. length (f h) = c
    and xs = concat (map f [0..<G])
    and j < G * c
  shows ∃i h. i < c ∧ h < G ∧ xs ! j = f h ! i
  using assms(2,3)
proof (induction G arbitrary: xs)
  case 0
  then show ?case
    by simp
next
  case (Suc G)
  then have *: xs = concat (map f [0..<G]) @ f G (is xs = ?ys @ f G)
    by simp
  show ?case
  proof (cases j < G * c)
  case True
    then have ∃i h. i < c ∧ h < G ∧ ?ys ! j = f h ! i
      using Suc by simp
    then have ∃i h. i < c ∧ h < Suc G ∧ ?ys ! j = f h ! i
      using less-SucI by blast
    then have ∃i h. i < c ∧ h < Suc G ∧ xs ! j = f h ! i
      using True * by (simp add: assms(1) length-concat-const nth-append)
    then show ?thesis .
  next
    case False
    then have j ≥ G * c
      by simp
    define h where h = G
    then have h: h < Suc G
      by simp
    define i where i = j − G * c
    then have i: i < c
      using False Suc.prems(2) by auto
```

    have *xs* ! *j* = *f h* ! *i*
      using *assms* **by** (*simp add*: ∗ *False h-def i-def length-concat-const nth-append*)
    then show *?thesis*
      using *h i* **by** *auto*
  qed
qed

The cartesian TM has one tape less than the original TM.

lemma *cartesian-num-tapes*:
  assumes *turing-machine* (*Suc k*) *G M*
    and *M′* = *cartesian M b*
    and *length rs* = *k*
    and *q′* < *length M′*
  shows *length* (*snd* ((*M′* ! *q′*) *rs*)) = *k*
proof −
  define *q* **where** *q* = *q′ mod length M*
  define *h* **where** *h* = *q′ div length M*
  then have *h* < *b q′* = *h* ∗ *length M* + *q*
    using *q-def assms*(*2*) *assms*(*4*) *length-cartesian less-mult-imp-div-less* **by** *auto*
  then have *q* < *length M*
    using *q-def assms*(*2*,*4*) *length-cartesian*
    by (*metis add-lessD1 length-0-conv length-greater-0-conv mod-less-divisor mult-0-right*)

  have (*M′* ! *q′*) *rs* =
    (*let* (*q′*, *as*) = (*M* ! *q*) (*rs* @ [(*h* + *1*) *mod b*])
     in (*if q′* = *length M then b* ∗ *length M else* (*fst* (*last as*) + *b* − *1*) *mod b* ∗ *length M* + *q′*,
        butlast as))
    using *cartesian-at*[*OF assms*(*2*) ‹*h* < *b*› ‹*q* < *length M*›] ‹*q′* = *h* ∗ *length M* + *q*› **by** *simp*
  then have *snd* ((*M′* ! *q′*) *rs*) = (*let* (*q′*, *as*) = (*M* ! *q*) (*rs* @ [(*h* + *1*) *mod b*]) *in butlast as*)
    by (*metis* (*no-types*, *lifting*) *case-prod-unfold snd-conv*)
  then have ∗: *snd* ((*M′* ! *q′*) *rs*) = *butlast* (*snd* ((*M* ! *q*) (*rs* @ [(*h* + *1*) *mod b*])))
    by (*simp add*: *case-prod-unfold*)

  have *length* (*rs* @ [(*h* + *1*) *mod b*]) = *Suc k*
    using *assms*(*3*) **by** *simp*
  then have *length* (*snd* ((*M* ! *q*) (*rs* @ [(*h* + *1*) *mod b*]))) = *Suc k*
    using *assms*(*1*) ‹*q* < *length M*› *turing-commandD*(*1*) *turing-machine-def nth-mem* **by** *metis*
  then show *?thesis*
    using ∗ **by** *simp*
qed

The cartesian TM of a TM with alphabet G also has the alphabet G provided it memorizes at most G values.

lemma *cartesian-tm*:
  assumes *turing-machine* (*Suc k*) *G M*
    and *M′* = *cartesian M b*
    and *k* ≥ *2*
    and *b* ≤ *G*
    and *b* > *0*
  shows *turing-machine k G M′*
proof
  show *G* ≥ *4*
    using *assms*(*1*) *turing-machine-def* **by** *simp*
  show *2* ≤ *k*
    using *assms*(*3*) **.**

  define *f* **where** *f* =
   (λ*h*. *map* (λ*i rs*.
          let (*q*, *as*) = (*M* ! *i*) (*rs* @ [(*h* + *1*) *mod b*])
          in (*if q* = *length M then b* ∗ *length M else* (*fst* (*last as*) + *b* − *1*) *mod b* ∗ *length M* + *q*,
            butlast as))
      [*0*..<*length M*])
  then have *1*: ⋀*h*. *length* (*f h*) = *length M*

```
  by simp
have 2: M' = concat (map f [0..<b])
  using f-def assms(2) cartesian-def by simp

show turing-command k (length M') G (M' ! j) if j < length M' for j
proof
  have 3: j < b * length M
    using that by (simp add: assms(2) length-cartesian)
  with 1 2 concat-nth-ex have ∃ i h. i < length M ∧ h < b ∧ M' ! j = f h ! i
    by blast
  then obtain i h where
    i: i < length M and
    h: h < b and
    cmd: M' ! j = (λrs.
        let (q, as) = (M ! i) (rs @ [(h + 1) mod b])
        in (if q = length M then b * length M else (fst (last as) + b − 1) mod b * length M + q,
            butlast as))
    using f-def by auto
  have (h + 1) mod b < b
    using h by auto
  then have modb: (h + 1) mod b < G
    using assms(4) by linarith

  have tc: turing-command (Suc k) (length M) G (M ! i)
    using i assms(1) turing-machine-def by simp

  show goal1: ⋀gs. length gs = k ⟹ length ([!!] (M' ! j) gs) = length gs
  proof −
    fix gs :: symbol list
    assume a: length gs = k
    let ?q = fst ((M ! i) (gs @ [(h + 1) mod b]))
    let ?as = snd ((M ! i) (gs @ [(h + 1) mod b]))
    have (M' ! j) gs =
        (if ?q = length M then b * length M else (fst (last ?as) + b − 1) mod b * length M + ?q,
        butlast ?as)
      using cmd by (metis (no-types, lifting) case-prod-unfold)
    then have [!!] (M' ! j) gs = butlast ?as
      by simp
    moreover have length ?as = Suc k
      using a turing-commandD(1)[OF tc] by simp
    ultimately show length ([!!] (M' ! j) gs) = length gs
      by (simp add: a)
  qed
  show (M' ! j) gs [.] ja < G
    if length gs = k and
      (⋀i. i < length gs ⟹ gs ! i < G) and
      ja < length gs
    for gs ja
  proof −
    let ?q = fst ((M ! i) (gs @ [(h + 1) mod b]))
    let ?as = snd ((M ! i) (gs @ [(h + 1) mod b]))
    have *: (M' ! j) gs =
        (if ?q = length M then b * length M else (fst (last ?as) + b − 1) mod b * length M + ?q,
        butlast ?as)
      using cmd by (metis (no-types, lifting) case-prod-unfold)
    have length (gs @ [(h + 1) mod b]) = Suc k
      using that by simp
    moreover have (gs @ [(h + 1) mod b]) ! i < G if i < length (gs @ [(h + 1) mod b]) for i
      using that by (metis ‹⋀i. i < length gs ⟹ gs ! i < G› modb
        length-append-singleton less-Suc-eq nth-append nth-append-length)
    ultimately have (∀ j<length (gs @ [(h + 1) mod b]). (M ! i) (gs @ [(h + 1) mod b]) [.] j < G)
      using that turing-commandD(2)[OF tc] by simp
    moreover have butlast ?as ! ja = ?as ! ja
```

```
        by (metis * goal1 nth-butlast snd-conv that(1) that(3))
      ultimately show ?thesis
        using * that(3) by auto
    qed
    show (M′ ! j) gs [.] 0 = gs ! 0 if length gs = k and 0 < k for gs
    proof −
      let ?q = fst ((M ! i) (gs @ [(h + 1) mod b]))
      let ?as = snd ((M ! i) (gs @ [(h + 1) mod b]))
      have *: (M′ ! j) gs =
          (if ?q = length M then b * length M else (fst (last ?as) + b − 1) mod b * length M + ?q,
            butlast ?as)
        using cmd by (metis (no-types, lifting) case-prod-unfold)
      have length (gs @ [(h + 1) mod b]) = Suc k
        using that by simp
      then have (M ! i) (gs @ [(h + 1) mod b]) [.] 0 = gs ! 0
        using that turing-commandD(3)[OF tc] by (simp add: nth-append)
      then show ?thesis
        using that * by (metis goal1 nth-butlast snd-conv)
    qed
    show [*] ((M′ ! j) gs) ≤ length M′ if length gs = k for gs
    proof −
      let ?q = [*] ((M ! i) (gs @ [(h + 1) mod b]))
      let ?as = snd ((M ! i) (gs @ [(h + 1) mod b]))
      have *: (M′ ! j) gs =
        (if ?q = length M then b * length M else (fst (last ?as) + b − 1) mod b * length M + ?q,
          butlast ?as)
        using cmd by (metis (no-types, lifting) case-prod-unfold)
      have length (gs @ [h]) = Suc k
        using that by simp
      then have ?q ≤ length M
        using assms(1) i turing-commandD(4)[OF tc] by (metis length-append-singleton)
      show ?thesis
      proof (cases ?q = length M)
        case True
        then show ?thesis
          using * by (simp add: assms(2) length-cartesian)
      next
        case False
        then have ?q < length M
          using ‹?q ≤ length M› by simp
        then have **: [*] ((M′ ! j) gs) = (fst (last ?as) + b − 1) mod b * length M + ?q
          using * by simp
        have (fst (last ?as) + b − 1) mod b ≤ b − 1
          using h less-imp-Suc-add by fastforce
        have (fst (last ?as) + b − 1) mod b * length M ≤ b * length M − length M
          using h less-imp-Suc-add by fastforce
        then have (fst (last ?as) + b − 1) mod b * length M + ?q ≤ b * length M − length M + ?q
          by simp
        then have (fst (last ?as) + b − 1) mod b * length M + ?q < b * length M
          using ‹?q < length M› 3 assms(5) by auto
        then show ?thesis
          using length-cartesian ** assms(2) by simp
      qed
    qed
  qed
qed
```

A special case of the previous lemma is $b = G$:

```
corollary cartesian-tm′:
  assumes turing-machine (Suc k) G M
    and M′ = cartesian M G
    and k ≥ 2
  shows turing-machine k G M′
```

using *assms cartesian-tm* **by** (*metis gr0I not-numeral-le-zero order-refl turing-machine-def*)

A cartesian TM assumes essentially the same configurations the original machine does, except that it has one tape less and the states have a greater number. We call these configurations "squished", another fancy made-up term alluding to the removal of one tape.

definition *squish* :: *nat* ⇒ *nat* ⇒ *config* ⇒ *config* **where**
squish G Q cfg ≡
let (*q*, *tps*) = *cfg*
in (*if q* ≥ *Q then G* ∗ *Q else* (|.| (*last tps*) + *G* − *1*) *mod G* ∗ *Q* + *q*, *butlast tps*)

lemma *squish*:
squish G Q cfg =
(*if fst cfg* ≥ *Q then G* ∗ *Q else* (|.| (*last* (*snd cfg*)) + *G* − *1*) *mod G* ∗ *Q* + *fst cfg*, *butlast* (*snd cfg*))
using *squish-def* **by** (*simp add*: *case-prod-beta*)

lemma *squish-head-pos*:
assumes ||*cfg*|| > *2*
shows *squish G Q cfg* <#> *0* = *cfg* <#> *0*
and *squish G Q cfg* <#> *1* = *cfg* <#> *1*
using *assms squish*
by (*metis One-nat-def Suc-1 Suc-lessD length-butlast less-diff-conv nth-butlast plus-1-eq-Suc snd-conv*,
metis One-nat-def Suc-1 length-butlast less-diff-conv nth-butlast plus-1-eq-Suc snd-conv)

lemma *mod-less*:
fixes *q Q h G* :: *nat*
assumes *q* < *Q* **and** *0* < *G*
shows *h mod G* ∗ *Q* + *q* < *G* ∗ *Q*
proof −
have *h mod G* ≤ *G* − *1*
using *assms*(*2*) *less-Suc-eq-le* **by** *fastforce*
then have *h mod G* ∗ *Q* ≤ (*G* − *1*) ∗ *Q*
by *simp*
then have *h mod G* ∗ *Q* ≤ *G* ∗ *Q* − *Q*
by (*simp add*: *left-diff-distrib′*)
then have *h mod G* ∗ *Q* + *q* ≤ *G* ∗ *Q* − *Q* + *q*
by *simp*
then have *h mod G* ∗ *Q* + *q* ≤ *G* ∗ *Q* − *1*
using *assms* **by** *simp*
then show *?thesis*
by (*metis One-nat-def Suc-pred add.left-neutral add.right-neutral add-mono-thms-linordered-semiring*(*1*)
assms le-simps(*2*) *linorder-not-less mult-pos-pos zero-le*)
qed

lemma *squish-halt-state*:
assumes *G* > *0* **and** *fst cfg* ≤ *Q*
shows *fst* (*squish G Q cfg*) = *G* ∗ *Q* ⟷ *fst cfg* = *Q*
proof
show *fst cfg* = *Q* ⟹ *fst* (*squish G Q cfg*) = *G* ∗ *Q*
by (*simp add*: *squish*)
show *fst* (*squish G Q cfg*) = *G* ∗ *Q* ⟹ *fst cfg* = *Q*
proof (*rule ccontr*)
assume *a*: *fst* (*squish G Q cfg*) = *G* ∗ *Q*
assume *fst cfg* ≠ *Q*
then have *fst cfg* < *Q*
using *assms*(*2*) **by** *simp*
then have *fst* (*squish G Q cfg*) = (|.| (*last* (*snd cfg*)) + *G* − *1*) *mod G* ∗ *Q* + *fst cfg*
using *squish* **by** *simp*
also have ... < *G* ∗ *Q*
using *mod-less*[*OF* ‹*fst cfg* < *Q*› *assms*(*1*)] **by** *simp*
finally have *fst* (*squish G Q cfg*) < *G* ∗ *Q* **.**
with *a* **show** *False*
by *simp*

  qed
qed

lemma *butlast-replicate*: *butlast* (*replicate k x*) = *replicate* (*k* − *Suc 0*) *x*
  by (*intro nth-equalityI*) (*simp-all add*: *nth-butlast*)

lemma *squish-start-config*: *G* ≥ *4* ⟹ *k* ≥ *2* ⟹ *squish G Q* (*start-config* (*Suc k*) *zs*) = *start-config k zs*
  using *squish-def start-config-def* **by** (*simp add*: *butlast-replicate*)

The cartesian Turing machine only works properly if the original TM never moves its head on the last tape. We call a tape of a TM M *immobile* if M never moves the head on the tape.

definition *immobile* :: *machine* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
 immobile M j k ≡ ∀ *q rs*. *q* < *length M* ⟶ *length rs* = *k* ⟶ (*M* ! *q*) *rs* [~] *j* = *Stay*

lemma *immobileI* [*intro*]:
  assumes ⋀*q rs*. *q* < *length M* ⟹ *length rs* = *k* ⟹ (*M* ! *q*) *rs* [~] *j* = *Stay*
  shows *immobile M j k*
  using *immobile-def assms* **by** *simp*

If the head never moves on tape k, the head will stay in position 0.

lemma *immobile-head-pos-proper*:
  assumes *proper-machine* (*Suc k*) *M*
    and *immobile M k* (*Suc k*)
    and ||*cfg*|| = *Suc k*
  shows *execute M cfg t* <#> *k* = *cfg* <#> *k*
proof (*induction t*)
  case *0*
  then show *?case*
    by *simp*
next
  case (*Suc t*)
  have *execute M cfg* (*Suc t*) = *exe M* (*execute M cfg t*)
    (**is** - = *exe M ?cfg*)
    by *simp*
  show *?case*
  proof (*cases fst ?cfg* ≥ *length M*)
    case *True*
    then have *exe M ?cfg* = *?cfg*
      using *exe-ge-length* **by** *simp*
    then show *?thesis*
      by (*simp add*: *Suc.IH*)
  next
    case *False*
    let *?cmd* = *M* ! (*fst ?cfg*)
    let *?rs* = *config-read ?cfg*
    have *exe M ?cfg* = *sem ?cmd ?cfg*
      using *False exe-def* **by** *simp*
    moreover have *proper-command* (*Suc k*) (*M* ! (*fst ?cfg*))
      using *assms*(*1*) *False* **by** *simp*
    ultimately have *exe M ?cfg* <!> *k* = *act* (*snd* (*?cmd ?rs*) ! *k*) (*?cfg* <!> *k*)
      using *assms execute-num-tapes-proper lessI sem-snd* **by** *presburger*
    then show *?thesis*
      using *False Suc act assms execute-num-tapes-proper immobile-def read-length* **by** *simp*
  qed
qed

lemma *immobile-head-pos*:
  assumes *turing-machine* (*Suc k*) *G M*
    and *immobile M k* (*Suc k*)
    and ||*cfg*|| = *Suc k*
  shows *execute M cfg t* <#> *k* = *cfg* <#> *k*
proof (*induction t*)
  case *0*

```
    then show ?case
      by simp
next
  case (Suc t)
  have execute M cfg (Suc t) = exe M (execute M cfg t)
    (is - = exe M ?cfg)
    by simp
  show ?case
  proof (cases fst ?cfg ≥ length M)
    case True
    then have exe M ?cfg = ?cfg
      using exe-ge-length by simp
    then show ?thesis
      by (simp add: Suc.IH)
  next
    case False
    let ?cmd = M ! (fst ?cfg)
    let ?rs = config-read ?cfg
    have exe M ?cfg = sem ?cmd ?cfg
      using False exe-def by simp
    moreover have proper-command (Suc k) (M ! (fst ?cfg))
      using assms(1) False by (metis turing-commandD(1) linorder-not-le turing-machineD(3))
    ultimately have exe M ?cfg <!> k = act (snd (?cmd ?rs) ! k) (?cfg <!> k)
      using assms execute-num-tapes lessI sem-snd by presburger
    then show ?thesis
      using False Suc act assms execute-num-tapes immobile-def read-length by simp
  qed
qed
```

Sequentially combining two Turing machines with an immobile tape yields a Turing machine with the same immobile tape.

```
lemma immobile-sequential:
  assumes turing-machine k G M1
    and turing-machine k G M2
    and immobile M1 j k
    and immobile M2 j k
  shows immobile (M1 ;; M2) j k
proof
  let ?M = M1 ;; M2
  fix q :: nat and rs :: symbol list
  assume q: q < length ?M and rs: length rs = k
  show (?M ! q) rs [~] j = Stay
  proof (cases q < length M1)
    case True
    then have ?M ! q = M1 ! q
      by (simp add: nth-append turing-machine-sequential-def)
    then show ?thesis
      using assms(3) immobile-def by (simp add: True rs)
  next
    case False
    then have ?M ! q = relocate-cmd (length M1) (M2 ! (q − length M1))
      using q turing-machine-sequential-nth' by simp
    then show ?thesis
      using relocate-cmd-head False assms(4) q rs length-turing-machine-sequential immobile-def
      by simp
  qed
qed
```

A loop also keeps a tape immobile.

```
lemma immobile-loop:
  assumes turing-machine k G M1
    and turing-machine k G M2
    and immobile M1 j k
```

```
    and immobile M2 j k
    and j < k
  shows immobile (WHILE M1 ; cond DO M2 DONE) j k
proof
  let ?loop = WHILE M1 ; cond DO M2 DONE
  have ?loop =
      M1 @
      [cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)] @
      (relocate (length M1 + 1) M2) @
      [cmd-jump (λ-. True) 0 0]
      (is - = M1 @ [?a] @ ?bs @ [?c])
    using turing-machine-loop-def by simp
  then have loop: ?loop = (M1 @ [?a]) @ (?bs @ [?c])
    by simp
  fix q :: nat
  assume q: q < length ?loop
  fix rs :: symbol list
  assume rs: length rs = k
  consider
      q < length M1
    | q = length M1
    | length M1 < q ∧ q ≤ length M1 + length M2
    | length M1 + length M2 < q
    by linarith
  then show (?loop ! q) rs [~] j = Stay
  proof (cases)
    case 1
    then have ?loop ! q = M1 ! q
      by (simp add: nth-append turing-machine-loop-def)
    then show ?thesis
      using assms(3) 1 rs immobile-def by simp
  next
    case 2
    then have ?loop ! q = cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)
      by (simp add: nth-append turing-machine-loop-def)
    then show ?thesis
      using rs cmd-jump-def assms(5) by simp
  next
    case 3
    then have ?loop ! q = (?bs @ [?c]) ! (q − (length M1 + 1))
      using nth-append[of M1 @ [?a] ?bs @ [?c]] loop by simp
    moreover have q − (length M1 + 1) < length ?bs
      using 3 length-relocate by auto
    ultimately have ?loop ! q = ?bs ! (q − (length M1 + 1))
      by (simp add: nth-append)
    then show ?thesis
      using assms(4,5) relocate-cmd-head 3 relocate rs immobile-def by auto
  next
    case 4
    then have q = length M1 + length M2 + 1
      using q turing-machine-loop-len by simp
    then have ?loop ! q = ?c
      using turing-machine-loop-def
      by (metis (no-types, lifting) One-nat-def Suc-eq-plus1 append-assoc length-append list.size(3)
        list.size(4) nth-append-length plus-nat.simps(2) length-relocate)
    then show ?thesis
      using rs cmd-jump-def assms(5) by simp
  qed
qed
```

An immobile tape stays immobile when further tapes are appended. We only need this for the special case of two-tape Turing machines.

lemma *immobile-append-tapes*:

assumes $j < k$ **and** $j > 1$ **and** $k \geq 2$ **and** *turing-machine 2 G M*
shows *immobile* (*append-tapes 2 k M*) *j k*
proof
let *?M* = *append-tapes 2 k M*
fix *q* :: *nat*
assume *q*: *q* < *length ?M*
fix *rs* :: *symbol list*
assume *rs*: *length rs* = *k*
have *q* < *length M*
using *assms q* **by** (*metis length-append-tapes*)
show (*?M* ! *q*) *rs* [~] *j* = *Stay*
proof −
have (*?M* ! *q*) *rs* =
(*fst* ((*M* ! *q*) (*take 2 rs*)), *snd* ((*M* ! *q*) (*take 2 rs*)) @ (*map* (λ*j*. (*rs* ! *j*, *Stay*)) [*2*..<*k*]))
using *append-tapes-nth* **by** (*simp add*: *append-tapes-nth* ‹*q* < *length M*› *rs*)
then have (*?M* ! *q*) *rs* [~] *j* =
snd ((*snd* ((*M* ! *q*) (*take 2 rs*)) @ (*map* (λ*j*. (*rs* ! *j*, *Stay*)) [*2*..<*k*])) ! *j*)
using *assms* **by** *simp*
also have ... = *snd* ((*map* (λ*j*. (*rs* ! *j*, *Stay*)) [*2*..<*k*]) ! (*j* − *2*))
proof −
have *length* (*take 2 rs*) = *2*
using *rs assms*(*3*) **by** *simp*
then have *length* ([!!] (*M* ! *q*) (*take 2 rs*)) = *2*
using *assms rs* **by** (*metis turing-commandD*(*1*) ‹*q* < *length M*› *nth-mem turing-machine-def*)
then show *?thesis*
using *nth-append*[*of snd* ((*M* ! *q*) (*take 2 rs*)) *map* (λ*j*. (*rs* ! *j*, *Stay*)) [*2*..<*k*] *j*] *assms* **by** *simp*
qed
finally show *?thesis*
using *assms*(*1*,*2*) **by** *simp*
qed
qed

For the elementary Turing machines we introduced in Section 2.4 all tapes are immobile but the ones given as parameters.

lemma *immobile-tm-trans-until*:
assumes $j \neq j1$ **and** $j \neq j2$ **and** $j < k$
shows *immobile* (*tm-trans-until j1 j2 H f*) *j k*
using *assms tm-trans-until-def cmd-trans-until-def* **by** *auto*

lemma *immobile-tm-ltrans-until*:
assumes $j \neq j1$ **and** $j \neq j2$ **and** $j < k$
shows *immobile* (*tm-ltrans-until j1 j2 H f*) *j k*
using *assms tm-ltrans-until-def cmd-ltrans-until-def* **by** *auto*

lemma *immobile-tm-left-until*:
assumes $j \neq j'$ **and** $j < k$
shows *immobile* (*tm-left-until H j'*) *j k*
using *assms tm-left-until-def cmd-left-until-def* **by** *auto*

lemma *immobile-tm-start*:
assumes $j \neq j'$ **and** $j < k$
shows *immobile* (*tm-start j'*) *j k*
using *tm-start-def immobile-tm-left-until*[*OF assms*] **by** *metis*

lemma *immobile-tm-write*:
assumes $j < k$
shows *immobile* (*tm-write j' h*) *j k*
using *assms tm-write-def cmd-write-def* **by** *auto*

lemma *immobile-tm-write-many*:
assumes $j < k$
shows *immobile* (*tm-write-many J h*) *j k*
using *assms tm-write-many-def cmd-write-many-def* **by** *auto*

lemma *immobile-tm-right*:
  assumes $j \neq j'$ **and** $j < k$
  shows *immobile* (*tm-right* j') j k
  using *assms tm-right-def cmd-right-def* **by** *auto*

lemma *immobile-tm-rtrans*:
  assumes $j \neq j'$ **and** $j < k$
  shows *immobile* (*tm-rtrans* j' f) j k
  using *assms tm-rtrans-def cmd-rtrans-def* **by** *auto*

lemma *immobile-tm-left*:
  assumes $j \neq j'$ **and** $j < k$
  shows *immobile* (*tm-left* j') j k
  using *assms tm-left-def cmd-left-def* **by** *auto*

lemma *mod-inc-dec*: $(h{::}nat) < G \Longrightarrow ((h + G - 1) \; mod \; G + 1) \; mod \; G = h$
  using *mod-Suc-eq* **by** *auto*

lemma *last-length*: $length \; xs = Suc \; k \Longrightarrow last \; xs = xs \; ! \; k$
  by (*metis diff-Suc-1 last-conv-nth length-0-conv nat.simps*(*3*))

The tapes used for memorizing the values have blank symbols in every cell but possibly for the leftmost cell. In keeping with funny names, we call such tapes *onesie* tapes.

definition *onesie* :: *symbol* ⇒ *tape* (‹⌈-⌉›) **where**
  $\lceil h \rceil \equiv (\lambda x. \; if \; x = 0 \; then \; h \; else \; \Box, \; 0)$

lemma *onesie-1*: $\lceil \triangleright \rceil = (\lfloor [] \rfloor, \; 0)$
  unfolding *onesie-def contents-def* **by** *auto*

lemma *onesie-read* [*simp*]: $|.| \; \lceil h \rceil = h$
  using *onesie-def* **by** *simp*

lemma *onesie-write*: $\lceil x \rceil \; |{:=}| \; y = \lceil y \rceil$
  using *onesie-def* **by** *auto*

lemma *act-onesie*: $act \; (h, \; Stay) \; \lceil x \rceil = \lceil h \rceil$
  using *onesie-def* **by** *auto*

We now consider the semantics of cartesian Turing machines. Roughly speaking, a cartesian TM assumes the squished configurations of the original TM. A crucial assumption here is that the original TM only ever memorizes a symbol from a certain range of symbols, with one relaxation: when switching to the halting state, any symbol may be written to the memorization tape. The reason is that there is only one halting state even for the cartesian TM, and thus the halting state is not subject to the mapping of states implemented by the *cartesian* operation.

In the following lemma, $\lceil \triangleright \rceil$ is the memorization tape. It has the start symbol because in the start configuration all tapes have the start symbol in the leftmost cell.

lemma *cartesian-execute*:
  assumes *turing-machine* (*Suc k*) *G M*
    and *immobile M k* (*Suc k*)
    and $k \geq 2$
    and $b > 0$
    and $length \; tps = k$
    and $\bigwedge t. \; execute \; M \; (0, \; tps \; @ \; [\lceil \triangleright \rceil]) \; t \; <.> \; k < b \lor fst \; (execute \; M \; (0, \; tps \; @ \; [\lceil \triangleright \rceil]) \; t) = length \; M$
  shows $execute \; (cartesian \; M \; b) \; (0, \; tps) \; t =$
    $squish \; b \; (length \; M) \; (execute \; M \; (0, \; tps \; @ \; [\lceil \triangleright \rceil]) \; t)$
proof (*induction t*)
  case *0*
  then show *?case*
    using *squish* **by** *simp*
next
  case (*Suc t*)

```
let ?M′ = cartesian M b
have len: length ?M′ = b ∗ length M
  using length-cartesian by simp
let ?cfg = execute M (0, tps @ [[▷]]) t
have 1: ?cfg <#> k = 0
  using assms(1,2,5) immobile-head-pos onesie-def by auto
have 3: ||?cfg|| = Suc k
  using assms(1,5) execute-num-tapes by auto
let ?Q = length M
let ?squish = squish b ?Q
let ?scfg = ?squish ?cfg
obtain q tps where qtps: ?cfg = (q, tps)
  by fastforce
have length tps = Suc k
  using 3 qtps by simp
then have last: last tps = tps ! k
  using assms(4) by (metis diff-Suc-1 last-conv-nth length-greater-0-conv zero-less-Suc)

have exe ?M′ ?scfg = ?squish (exe M ?cfg)
proof (cases q ≥ length M)
  case True
  then have t1: exe M ?cfg = ?cfg
    using exe-def qtps by auto
  have ?scfg = (b ∗ length M, butlast tps)
    using True squish qtps by simp
  then have exe ?M′ ?scfg = ?scfg
    using exe-def len by simp
  with t1 show ?thesis
    by simp
next
  case False
  then have scfg: ?scfg = (( |.| (last tps) + b − 1) mod b ∗ length M + q, butlast tps)
      (is - = (?q, -))
    using squish qtps by simp
  moreover have q-less: ?q < length ?M′
    using mod-less False length-cartesian assms(4) by simp
  ultimately have 9: exe ?M′ ?scfg = sem (?M′ ! ?q) ?scfg
    using exe-def by simp
  let ?cmd′ = ?M′ ! ?q
  let ?h = ( |.| (last tps) + b − 1) mod b
  have q < length M
    using False by simp
  then have 4: |.| (last tps) < b
    using assms last qtps False by (metis fst-conv less-not-refl3 snd-conv)
  have h-less: ?h < b
    using 4 by simp
  then have ?cmd′ ≡ λrs.
      (let (q′, as) = (M ! q) (rs @ [(?h + 1) mod b])
       in (if q′ = length M then b ∗ length M else (fst (last as) + b − 1) mod b ∗ length M + q′,
          butlast as))
    using cartesian-at[OF - h-less ‹q < length M›] by presburger
  then have cmd′: ?cmd′ ≡ λrs.
      (let (q′, as) = (M ! q) (rs @ [ |.| (last tps)])
       in (if q′ = length M then b ∗ length M else (fst (last as) + b − 1) mod b ∗ length M + q′,
          butlast as))
    using mod-inc-dec[OF 4] by simp
  let ?rs′ = config-read ?scfg
  have 10: sem ?cmd′ ?scfg =
      (let (newstate, as) = ?cmd′ ?rs′
       in (newstate, map (λ(a, tp). act a tp) (zip as (butlast tps))))
    using scfg by (simp add: sem-def)
  let ?newstate′ = fst (?cmd′ ?rs′)
  let ?as′ = snd (?cmd′ ?rs′)
```

```
have 11: sem ?cmd' ?scfg =
    (?newstate', map (λ(a, tp). act a tp) (zip ?as' (butlast tps)))
  using 10 by (simp add: case-prod-beta)
with 9 have lhs: exe ?M' ?scfg =
    (?newstate', map (λ(a, tp). act a tp) (zip ?as' (butlast tps)))
  by simp

let ?cmd = M ! q
let ?rs = config-read ?cfg
let ?newstate = fst (?cmd ?rs)
let ?as = snd (?cmd ?rs)
have ?squish (exe M ?cfg) = ?squish (sem ?cmd ?cfg)
  using qtps False exe-def by simp
also have ... = ?squish (?newstate, map (λ(a, tp). act a tp) (zip ?as (snd ?cfg)))
  by (metis sem)
also have ... = ?squish (?newstate, map (λ(a, tp). act a tp) (zip ?as tps))
   (is - = ?squish (-, ?tpsSuc))
  using qtps by simp
also have ... =
  (if ?newstate ≥ ?Q then b ∗ ?Q else ( |.| (last ?tpsSuc) + b − 1) mod b ∗ ?Q + ?newstate,
   butlast ?tpsSuc)
  using squish by simp
finally have rhs: ?squish (exe M ?cfg) =
  (if ?newstate ≥ ?Q then b ∗ ?Q else ( |.| (last ?tpsSuc) + b − 1) mod b ∗ ?Q + ?newstate,
   butlast ?tpsSuc) .

have read (butlast tps) @ [ |.| (last tps)] = read tps
  using ‹length tps = Suc k› read-def
 by (metis (no-types, lifting) last-map length-0-conv map-butlast map-is-Nil-conv old.nat.distinct(1) snoc-eq-iff-butlast)
then have rs'-read: ?rs' @ [ |.| (last tps)] = read tps
  using scfg by simp

have fst: fst (?cmd' ?rs') =
  (if ?newstate ≥ ?Q then b ∗ ?Q else ( |.| (last ?tpsSuc) + b − 1) mod b ∗ ?Q + ?newstate)
proof −
  have fst (?cmd' ?rs') ≡ fst (
    (let (q', as) = (M ! q) (?rs' @ [ |.| (last tps)])
     in (if q' = length M then b ∗ length M else (fst (last as) + b − 1) mod b ∗ length M + q',
         butlast as)))
    using cmd' by simp
  then have fst (?cmd' ?rs') =
     (if fst ((M ! q) (?rs' @ [ |.| (last tps)])) = length M
      then b ∗ length M
      else (fst (last (snd ((M ! q) (?rs' @ [ |.| (last tps)])))) + b − 1) mod b ∗ length M +
        fst ((M ! q) (?rs' @ [ |.| (last tps)])))
    by (auto simp add: Let-def split-beta)
  then have lhs: fst (?cmd' ?rs') =
     (if fst ((M ! q) (read tps)) = length M
      then b ∗ ?Q
      else (fst (last (snd ((M ! q) (read tps)))) + b − 1) mod b ∗ ?Q + fst ((M ! q) (read tps)))
    using rs'-read by simp

  have ∗: |.| (last (map (λ(a, tp). act a tp) (zip ?as tps))) = fst (last (snd ((M ! q) (read tps))))
  proof −
    have length ?as = Suc k
      using 3 ‹q < length M› assms(1) read-length turing-machine-def
      by (metis turing-commandD(1) nth-mem)
    then have length (map (λ(a, tp). act a tp) (zip ?as tps)) = Suc k
      using ‹length tps = Suc k› by simp
    then have last (map (λ(a, tp). act a tp) (zip ?as tps)) =
        (map (λ(a, tp). act a tp) (zip ?as tps)) ! k
      using last-length by blast
    moreover have proper-command (Suc k) ?cmd
```

using ‹q < *length* M› *assms*(*1*) *turing-machine-def turing-commandD*(*1*) *nth-mem* **by** *blast*
ultimately have *1*: *last* (*map* (λ(*a*, *tp*). *act a tp*) (*zip ?as tps*)) = *act* (*?as* ! *k*) (*tps* ! *k*)
using *3* ‹q < *length* M› *assms*(*1*) *qtps sem′ sem-snd-tm* **by** *auto*

have *snd* (*?as* ! *k*) = *Stay*
using *assms*(*2*) ‹*length tps* = *Suc k*› *3* ‹q < *length* M› *read-length immobile-def*
by *simp*
then have *act* (*?as* ! *k*) (*tps* ! *k*) = (*tps* ! *k*) |:=| (*fst* (*?as* ! *k*))
by (*metis act-Stay′ prod.collapse*)
then have |.| (*act* (*?as* ! *k*) (*tps* ! *k*)) = *fst* (*?as* ! *k*)
by *simp*
with *1* **have** *2*: |.| (*last* (*map* (λ(*a*, *tp*). *act a tp*) (*zip ?as tps*))) = *fst* (*?as* ! *k*)
by *simp*

have *fst* (*last* (*snd* ((*M* ! *q*) (*read tps*)))) = *fst* (*last ?as*)
using *qtps* **by** *simp*
then have *fst* (*last* (*snd* ((*M* ! *q*) (*read tps*)))) = *fst* (*?as* ! *k*)
using ‹*length ?as* = *Suc k*› **by** (*simp add*: *last-length*)
with *2* **show** *?thesis*
by *simp*
qed

have (*if fst* ((*M* ! *q*) *?rs*) ≥ *?Q*
then b ∗ *?Q*
else (|.| (*last ?tpsSuc*) + *b* − *1*) *mod b* ∗ *?Q* + *fst* ((*M* ! *q*) *?rs*)) =
(*if fst* ((*M* ! *q*) (*read tps*)) ≥ *?Q*
then b ∗ *?Q*
else (|.| (*last ?tpsSuc*) + *b* − *1*) *mod b* ∗ *?Q* + *fst* ((*M* ! *q*) (*read tps*)))
using *qtps* **by** *simp*
also have ... =
(*if fst* ((*M* ! *q*) (*read tps*)) = *?Q*
then b ∗ *?Q*
else (|.| (*last ?tpsSuc*) + *b* − *1*) *mod b* ∗ *?Q* + *fst* ((*M* ! *q*) (*read tps*)))
using *assms*(*1*) *turing-machine-def*
by (*metis* (*mono-tags*, *lifting*) *3 turing-commandD*(*4*) ‹q < *length* M› *le-antisym nth-mem prod.sel*(*2*) *qtps read-length*)
also have ... =
(*if fst* ((*M* ! *q*) (*read tps*)) = *length M*
then b ∗ *?Q*
else (*fst* (*last* (*snd* ((*M* ! *q*) (*read tps*)))) + *b* − *1*) *mod b* ∗ *?Q* + *fst* ((*M* ! *q*) (*read tps*)))
using ∗ **by** *simp*
finally show *?thesis*
using *lhs* **by** *simp*
qed

have *snd*: *map* (λ(*a*, *tp*). *act a tp*) (*zip ?as′* (*butlast tps*)) =
butlast (*map* (λ(*a*, *tp*). *act a tp*) (*zip ?as tps*))
proof (*rule nth-equalityI*)
let *?lhs* = *map* (λ(*a*, *tp*). *act a tp*) (*zip ?as′* (*butlast tps*))
let *?rhs* = *butlast* (*map* (λ(*a*, *tp*). *act a tp*) (*zip ?as tps*))
have ||*?scfg*|| = *k*
using ‹*length tps* = *Suc k*› *scfg* **by** *simp*
then have *length ?rs′* = *k*
by (*simp add*: *read-length*)
then have *length ?as′* = *k*
using *cartesian-num-tapes q-less assms*(*1*,*3*) **by** *simp*
moreover have *length* (*butlast tps*) = *k*
using ‹*length tps* = *Suc k*› **by** *simp*
ultimately have *length ?lhs* = *k*
by *simp*

have *length ?as* = *Suc k*
using ‹*length tps* = *Suc k*› ‹q < *length* M› *assms*(*1*) *qtps read-length turing-machine-def*

```
      by (metis 3 turing-commandD(1) nth-mem)
    then have length ?rhs = k
      using ‹length tps = Suc k› by simp
    then show length ?lhs = length ?rhs
      using ‹length ?lhs = k› by simp

    show ?lhs ! j = ?rhs ! j if j < length ?lhs for j
    proof -
      have j < k
        using that ‹length ?lhs = k› by auto

      have length (butlast tps) = k
        using ‹length (butlast tps) = k› by blast
      then have lhs: ?lhs ! j = act (?as' ! j) (tps ! j)
        using ‹j < k› ‹length ?as' = k› by (simp add: nth-butlast)

      have rhs: ?rhs ! j = act (?as ! j) (tps ! j)
        using ‹j < k› ‹length ?as = Suc k› ‹length tps = Suc k› by (simp add: nth-butlast)

      have ?as' ! j = snd (
          (let (q', as) = (M ! q) (?rs' @ [ |.| (last tps)])
           in (if q' = length M then b * length M else (fst (last as) + b − 1) mod b * length M + q',
               butlast as))) ! j
        using cmd' by simp
      also have ... = snd (
          ((if fst ((M ! q) (?rs' @ [ |.| (last tps)])) = length M
            then b * length M
             else (fst (last (snd ((M ! q) (?rs' @ [ |.| (last tps)])))) + b − 1) mod b * length M + fst ((M ! q)
(?rs' @ [ |.| (last tps)])),
            butlast (snd ((M ! q) (?rs' @ [ |.| (last tps)]))))) ! j
        by (metis (no-types, lifting) case-prod-unfold)
      also have ... = butlast (snd ((M ! q) (?rs' @ [ |.| (last tps)]))) ! j
        by simp
      finally have ?as' ! j = butlast (snd ((M ! q) (?rs' @ [ |.| (last tps)]))) ! j .
      then have as'-j: ?as' ! j = butlast (snd ((M ! q) (read tps))) ! j
        using rs'-read by simp

      have ?as ! j = snd ((M ! q) (read tps)) ! j
        using qtps by simp
      moreover have ?as ! j = (butlast ?as) ! j
        using ‹length ?as = Suc k› ‹j < k› by (simp add: nth-butlast)
      ultimately have ?as ! j = butlast (snd ((M ! q) (read tps))) ! j
        using qtps by simp
      then have ?as ! j = ?as' ! j
        using as'-j by simp
      then show ?thesis
        using lhs rhs by simp
    qed
  qed
  then show ?thesis
    using fst lhs rhs by simp
 qed
 then show ?case
   by (simp add: Suc.IH)
qed
```

One assumption of the previous lemma is that the memorization tape can only contain a symbol from a certain range (except in the halting state). One way to achieve this is for the Turing machine to only ever write a symbol from that range to the memorization tape (or switch to the halting state). Formally:

```
definition bounded-write :: machine ⇒ nat ⇒ nat ⇒ bool where
  bounded-write M k b ≡
    ∀ q rs. q < length M ⟶ length rs = Suc k ⟶ (M ! q) rs [.] k < b ∨ fst ((M ! q) rs) = length M
```

The advantage of *bounded-write* is that it is a relatively easy to prove property of a Turing machine. With *bounded-write* the previous lemma, *cartesian-execute*, turns into the following one, where the assumption $b > 0$ becomes $b > 1$ because initially the memorization tape has the start symbol, represented by the number 1.

lemma *cartesian-execute-onesie*:
  assumes *turing-machine* (*Suc k*) *G M*
    and *immobile M k* (*Suc k*)
    and $k \geq 2$
    and $b > 1$
    and *length tps* = *k*
    and *bounded-write M k b*
  shows *execute* (*cartesian M b*) (*0*, *tps*) *t* = *squish b* (*length M*) (*execute M* (*0*, *tps* @ [⌈▷⌉]) *t*)
proof −
  have *execute M* (*0*, *tps* @ [⌈▷⌉]) *t* <.> *k* < *b* ∨ *fst* (*execute M* (*0*, *tps* @ [⌈▷⌉]) *t*) = *length M*
    for *t*
  proof (*induction t*)
    case *0*
    then show *?case*
      using *assms* **by** *auto*
  next
    case (*Suc t*)
    let *?tps* = *tps* @ [⌈▷⌉]
    have ∗: *execute M* (*0*, *?tps*) (*Suc t*) = *exe M* (*execute M* (*0*, *?tps*) *t*)
        (**is** - = *exe M ?cfg*)
      by *simp*
    show *?case*
    proof (*cases fst ?cfg* ≥ *length M*)
      case *True*
      then show *?thesis*
        using ∗ *Suc exe-ge-length* **by** *presburger*
    next
      case *False*
      let *?rs* = *config-read ?cfg*
      let *?q* = *fst ?cfg*
      let *?tps* = *snd ?cfg*
      have *len*: *length ?tps* = *Suc k*
        using *assms*(*1*,*5*) *execute-num-tapes* **by** *simp*
      have *pos*: *?tps* :#: *k* = *0*
        using *assms immobile-head-pos onesie-def* **by** *auto*
      have *lenrs*: *length ?rs* = *Suc k*
        using *len read-length* **by** *simp*
      have ∗∗: *exe M ?cfg* = *sem* (*M* ! *?q*) *?cfg*
        using *False exe-lt-length* **by** *simp*
      have ∗∗∗: *sem* (*M* ! *?q*) *?cfg* <!> *k* = *act* ((*M* ! *?q*) *?rs* [!] *k*) (*?tps* ! *k*)
      proof −
        have *proper-command* (*Suc k*) (*M* ! *?q*)
          by (*metis False turing-commandD*(*1*) *assms*(*1*) *le-less-linear nth-mem turing-machine-def*)
        then show *?thesis*
          using *len sem-snd* **by** *blast*
      qed

      have (*M* ! *?q*) *?rs* [~] *k* = *Stay*
        using *assms*(*2*) *lenrs False immobile-def* **by** *simp*
      then have *act*: *act* ((*M* ! *?q*) *?rs* [!] *k*) (*?tps* ! *k*) = (*?tps* ! *k*) |:=| ((*M* ! *?q*) *?rs* [.] *k*)
        using *act-Stay′* **by** (*metis prod.collapse*)
       show *?thesis*
      proof (*cases* (*M* ! *?q*) *?rs* [.] *k* < *b*)
        case *True*
        then have *sem* (*M* ! *?q*) *?cfg* <.> *k* < *b*
          using *pos* ∗∗∗ *act* **by** *simp*
        then show *?thesis*
          using ∗ ∗∗ **by** *simp*
      next

case *halt*: *False*
then have *fst* ((*M* ! *?q*) *?rs*) = *length M*
using *assms*(*6*) *bounded-write-def lenrs False le-less-linear* **by** *blast*
then show *?thesis*
using ∗ ∗∗ *sem-fst* **by** *simp*
qed
qed
qed
then show *?thesis*
using *cartesian-execute*[*OF assms*(*1*−*3*) - *assms*(*5*)] *assms*(*4*) **by** *simp*
qed

In the following lemma, the term ⌈*c*⌉ reflects the fact that in the halting state the memorized symbol can be anything.

lemma *cartesian-transforms-onesie*:
assumes *turing-machine* (*Suc k*) *G M*
and *immobile M k* (*Suc k*)
and $k \geq 2$
and $b > 1$
and *bounded-write M k b*
and *length tps* = *k*
and *transforms M* (*tps* @ [⌈▷⌉]) *t* (*tps′* @ [⌈*c*⌉])
shows *transforms* (*cartesian M b*) *tps t tps′*
proof −
have *execute M* (*0*, *tps* @ [⌈▷⌉]) *t* = (*length M*, *tps′* @ [⌈*c*⌉])
using *transforms-def transits-def* **by** (*metis* (*no-types*, *lifting*) *assms*(*7*) *execute-after-halting-ge fst-conv*)
then have *execute* (*cartesian M b*) (*0*, *tps*) *t* = *squish b* (*length M*) (*length M*, *tps′* @ [⌈*c*⌉])
using *assms cartesian-execute-onesie* **by** *simp*
moreover from *this* **have** *fst* (*execute* (*cartesian M b*) (*0*, *tps*) *t*) = *b* ∗ *length M*
using *squish-halt-state*[*of b - length M*] *One-nat-def assms*(*4*) **by** *simp*
ultimately have *execute* (*cartesian M b*) (*0*, *tps*) *t* = (*b* ∗ *length M*, *tps′*)
using *squish* **by** *simp*
then show *?thesis*
using *transforms-def transits-def length-cartesian* **by** *auto*
qed

A Turing machine with alphabet G, when started on a symbol sequence over G, is guaranteed to only write symbols from G to any of its tapes, including any memorization tapes. Therefore the last assumption of lemma *cartesian-execute* is satisfied. So in the case of the start configuration we do not need any extra assumptions such as *bounded-write*. This is formalized in the next lemma. The downside is that it can only be applied to "finished" TMs but not to reusable TMs, because these do not usually start in the start state.

lemma *cartesian-execute-start-config*:
assumes *turing-machine* (*Suc k*) *G M*
and *immobile M k* (*Suc k*)
and $k \geq 2$
and ∀ *i*<*length zs*. *zs* ! *i* < *G*
shows *execute* (*cartesian M G*) (*start-config k zs*) *t* =
squish G (*length M*) (*execute M* (*start-config* (*Suc k*) *zs*) *t*)
proof −
let *?tps* = *snd* (*start-config k zs*)
have *snd* (*start-config* (*Suc k*) *zs*) =
(λ*i*. *if i* = *0 then 1 else if i* ≤ *length zs then zs* ! (*i* − *1*) *else 0*, *0*) #
replicate k (λ*i*. *if i* = *0 then 1 else 0*, *0*)
using *start-config-def* **by** *auto*
also have ... = (λ*i*. *if i* = *0 then 1 else if i* ≤ *length zs then zs* ! (*i* − *1*) *else 0*, *0*) #
replicate (*k* − *1*) (λ*i*. *if i* = *0 then 1 else 0*, *0*) @ [(λ*i*. *if i* = *0 then 1 else 0*, *0*)]
using *assms*(*3*)
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-le-D Suc-pred less-Suc-eq-0-disj replicate-Suc replicate-append-same*)
also have ... = *snd* (*start-config k zs*) @ [(λ*i*. *if i* = *0 then 1 else 0*, *0*)]
using *start-config-def* **by** *auto*
finally have *snd* (*start-config* (*Suc k*) *zs*) = *snd* (*start-config k zs*) @ [⌈▷⌉]
using *onesie-def* **by** *auto*

then have ∗: *start-config* (*Suc k*) *zs* = (*0*, *?tps* @ [[⌈▷]])
  using *start-config-def* **by** *simp*
then have *execute M* (*0*, *?tps* @ [[⌈▷]]) *t* <.> *k* < *G* **for** *t*
  using *assms*(*1*,*4*) **by** (*metis lessI tape-alphabet*)
moreover have *G* ≥ *2*
  using *assms*(*1*) *turing-machine-def* **by** *simp*
moreover have *length ?tps* = *k*
  using *start-config-length assms*(*3*) **by** *simp*
ultimately have *execute* (*cartesian M G*) (*0*, *?tps*) *t* =
    squish G (*length M*) (*execute M* (*0*, *?tps* @ [[⌈▷]]) *t*)
  using *cartesian-execute*[*OF assms*(*1*−*3*)] **by** *simp*
moreover have *start-config k zs* = (*0*, *?tps*)
  using *start-config-def* **by** *simp*
ultimately show *?thesis*
  using ∗ **by** *simp*
qed

So far we have only considered single memorization tapes. But of course we can have more than one by iterating the *cartesian* function. Applying this functions once removes the final memorization tape, but leaves others intact, that is, it maintains immobile tapes:

lemma *cartesian-immobile*:
  assumes *turing-machine* (*Suc k*) *G M*
    and *j* < *k*
    and *immobile M j* (*Suc k*)
    and *M′* = *cartesian M G*
  shows *immobile M′ j k*
proof *standard+*
  fix *q* :: *nat* **and** *rs* :: *symbol list*
  assume *q*: *q* < *length M′* **and** *rs*: *length rs* = *k*
  have *q* < *G* ∗ *length M*
    using *assms*(*1*,*4*) *q length-cartesian* **by** *simp*
  then have *G* > *0*
    using *gr0I* **by** *fastforce*
  have *length M* > *0*
    using ‹*q* < *G* ∗ *length M*› **by** *auto*
  define *h* **where** *h* = *q div length M*
  moreover define *i* **where** *i* = *q mod length M*
  then have *i* < *length M*
    using ‹*0* < *length M*› *mod-less-divisor* **by** *simp*
  have *h* < *G*
    using *i-def h-def* ‹*q* < *G* ∗ *length M*› *less-mult-imp-div-less* **by** *blast*
  have *q* = *h* ∗ *length M* + *i*
    using *h-def i-def* **by** *simp*
  then have *M′* ! *q* = (λ*rs*.
      (*let* (*q′*, *as*) = (*M* ! *i*) (*rs* @ [(*h* + *1*) *mod G*])
       in (*if q′* = *length M then G* ∗ *length M else* (*fst* (*last as*) + *G* − *1*) *mod G* ∗ *length M* + *q′*,
          butlast as)))
    using *assms*(*1*,*4*) ‹*h* < *G*› ‹*i* < *length M*› *cartesian-at* **by** *auto*
  then have (*M′* ! *q*) *rs* =
      (*let* (*q′*, *as*) = (*M* ! *i*) (*rs* @ [(*h* + *1*) *mod G*])
       in (*if q′* = *length M then G* ∗ *length M else* (*fst* (*last as*) + *G* − *1*) *mod G* ∗ *length M* + *q′*,
          butlast as)) **for** *rs*
    by *simp*
  then have (*M′* ! *q*) *rs* =
      (*let qas* = (*M* ! *i*) (*rs* @ [(*h* + *1*) *mod G*])
       in (*if fst qas* = *length M then G* ∗ *length M else* (*fst* (*last* (*snd qas*)) + *G* − *1*) *mod G* ∗ *length M* + *fst*
qas,
          butlast (*snd qas*))) **for** *rs*
    by (*metis* (*no-types*, *lifting*) *old.prod.case prod.collapse*)
  then have (*M′* ! *q*) *rs* =
      (*if* (*fst* ((*M* ! *i*) (*rs* @ [(*h* + *1*) *mod G*]))) = *length M*
       then G ∗ *length M*
       else (*fst* (*last* (*snd* ((*M* ! *i*) (*rs* @ [(*h* + *1*) *mod G*])))) + *G* − *1*) *mod G* ∗ *length M* + *fst* ((*M* ! *i*) (*rs* @

```
[(h + 1) mod G])),
       butlast (snd ((M ! i) (rs @ [(h + 1) mod G])))) for rs
    by metis
  then have 1: snd ((M' ! q) rs) = butlast (snd ((M ! i) (rs @ [(h + 1) mod G]))) for rs
    by simp

  have len: length (rs @ [(h + 1) mod G]) = Suc k
    by (simp add: rs)
  then have 2: (M ! i) (rs @ [(h + 1) mod G]) [~] j = Stay
    using immobile-def assms(3) len ‹i < length M› by blast
  have length (snd ((M ! i) (rs @ [(h + 1) mod G]))) = Suc k
    using len assms(1) turing-machine-def by (metis turing-commandD(1) ‹i < length M› turing-machineD(3))
  then have butlast (snd ((M ! i) (rs @ [(h + 1) mod G]))) ! j =
      snd ((M ! i) (rs @ [(h + 1) mod G])) ! j
    using assms(2) by (simp add: nth-butlast)
  then have snd ((M' ! q) rs) ! j = snd ((M ! i) (rs @ [(h + 1) mod G])) ! j
    using 1 by simp
  then show (M' ! q) rs [~] j = Stay
    using 2 by simp
qed
```

With the next function, *icartesian*, we can strip several memorization tapes off.

```
fun icartesian :: nat ⇒ machine ⇒ nat ⇒ machine where
  icartesian 0 M G = M |
  icartesian (Suc k) M G = icartesian k (cartesian M G) G
```

Applying *icartesian* maintains the property of being a Turing machine. We show that only for the special case that all tapes but the input and output tapes are memorization tapes. In this case, we end up with a two-tape machine.

```
lemma icartesian-tm:
  assumes turing-machine (k + 2) G M
    and ⋀j. j < k ⟹ immobile M (j + 2) (k + 2)
  shows turing-machine 2 G (icartesian k M G)
  using assms(1,2)
proof (induction k arbitrary: M)
  case 0
  then show ?case
    by (metis add.left-neutral icartesian.simps(1))
next
  case (Suc k)
  let ?M = cartesian M G
  have turing-machine (Suc (k + 2)) G M
    using Suc by simp
  moreover have k + 2 ≥ 2
    by simp
  ultimately have turing-machine (k + 2) G ?M
    using ‹turing-machine (Suc (k + 2)) G M› cartesian-tm' by blast
  moreover have ⋀j. j < k ⟹ immobile ?M (j + 2) (k + 2)
    using cartesian-immobile Suc by simp
  ultimately have turing-machine 2 G (icartesian k ?M G)
    using Suc by simp
  then show turing-machine 2 G (icartesian (Suc k) M G)
    by simp
qed
```

At this point we ought to prove something about the semantics of *icartesian*. However, we will only need one specific result, which we can only express at the end of Section 5.1 after we have introduced oblivious Turing machines.

end

2.6 Composing functions

theory *Composing*
  imports *Elementary*
begin

For a Turing machine M_1 computing f_1 in time T_1 and a TM M_2 computing f_2 in time T_2 there is a TM M computing $f_2 \circ f_1$ in time $O(T_1(n) + \max_{m \le T_1(n)} T_2(m))$. If T_1 is monotone the time bound is $O(T_1 + T_2 \circ T_1)$; if T_1 and T_2 are polynomially bounded the running-time of M is polynomially bounded, too.

The Turing machines M_1 and M_2 can have both a different alphabet and number of tapes, so generally they cannot be composed by the ;; operator. To get around this we enlarge the alphabet and prepend and append tapes, so M has as many tapes as M_1 and M_2 combined. The following function returns the combined Turing machine M.

definition *compose-machines k1 G1 M1 k2 G2 M2* ≡
  enlarged G1 (*append-tapes k1* (*k1* + *k2*) *M1*) ;;
  tm-start 1 ;;
  tm-cp-until 1 k1 {□} ;;
  tm-erase-cr 1 ;;
  tm-start k1 ;;
  prepend-tapes k1 (*enlarged G2 M2*) ;;
  tm-cr (*k1* + *1*) ;;
  tm-cp-until (*k1* + *1*) *1* {□}

locale *compose* =
  fixes *k1 k2 G1 G2* :: *nat*
    and *M1 M2* :: *machine*
    and *T1 T2* :: *nat* ⇒ *nat*
    and *f1 f2* :: *string* ⇒ *string*
  assumes *tm-M1*: *turing-machine k1 G1 M1*
    and *tm-M2*: *turing-machine k2 G2 M2*
    and *computes1*: *computes-in-time k1 M1 f1 T1*
    and *computes2*: *computes-in-time k2 M2 f2 T2*
begin

definition *tm1* ≡ *enlarged G1* (*append-tapes k1* (*k1* + *k2*) *M1*)
definition *tm2* ≡ *tm1* ;; *tm-start 1*
definition *tm3* ≡ *tm2* ;; *tm-cp-until 1 k1* {□}
definition *tm4* ≡ *tm3* ;; *tm-erase-cr 1*
definition *tm5* ≡ *tm4* ;; *tm-start k1*
definition *tm56* ≡ *prepend-tapes k1* (*enlarged G2 M2*)
definition *tm6* ≡ *tm5* ;; *tm56*
definition *tm7* ≡ *tm6* ;; *tm-cr* (*k1* + *1*)
definition *tm8* ≡ *tm7* ;; *tm-cp-until* (*k1* + *1*) *1* {□}

definition *G* :: *nat* **where**
  G ≡ *max G1 G2*

lemma *G1*: *G1* ≤ *G* **and** *G2*: *G2* ≤ *G*
  using *G-def* **by** *simp-all*

lemma *k-ge*: *k1* ≥ *2 k2* ≥ *2*
  using *tm-M1 tm-M2 turing-machine-def* **by** *simp-all*

lemma *tm1-tm*: *turing-machine* (*k1* + *k2*) *G tm1*
  unfolding *tm1-def* **using** *turing-machine-enlarged append-tapes-tm tm-M1 G1* **by** *simp*

lemma *tm2-tm*: *turing-machine* (*k1* + *k2*) *G tm2*
  unfolding *tm2-def* **using** *tm1-tm tm-start-tm turing-machine-def* **by** *blast*

lemma *tm3-tm*: *turing-machine* (*k1* + *k2*) *G tm3*
  unfolding *tm3-def*

using *tm2-tm tm-cp-until-tm turing-machine-def k-ge turing-machine-sequential-turing-machine*
by (*metis add-leD1 less-add-same-cancel1 less-le-trans less-numeral-extra*(*1*) *nat-1-add-1*)

lemma *tm4-tm*: *turing-machine* (*k1* + *k2*) *G tm4*
unfolding *tm4-def*
using *tm3-tm tm-erase-cr-tm turing-machine-def turing-machine-sequential-turing-machine*
by (*metis Suc-1 Suc-le-lessD tm-erase-cr-tm zero-less-one*)

lemma *tm5-tm*: *turing-machine* (*k1* + *k2*) *G tm5*
unfolding *tm5-def*
using *tm4-tm tm-start-tm turing-machine-def turing-machine-sequential-turing-machine*
by *auto*

lemma *tm6-tm*: *turing-machine* (*k1* + *k2*) *G tm6*
unfolding *tm6-def*
using *tm5-tm tm56-def turing-machine-enlarged prepend-tapes-tm tm-M2 G2*
by *simp*

lemma *tm7-tm*: *turing-machine* (*k1* + *k2*) *G tm7*
unfolding *tm7-def* **using** *tm6-tm tm-cr-tm turing-machine-def* **by** *blast*

lemma *tm8-tm*: *turing-machine* (*k1* + *k2*) *G tm8*
unfolding *tm8-def*
using *tm7-tm tm-cp-until-tm turing-machine-def turing-machine-sequential-turing-machine k-ge*(*2*)
by (*metis add.commute add-less-cancel-right add-strict-increasing nat-1-add-1*
verit-comp-simplify1(*3*) *zero-less-one*)

context
fixes *x* :: *string*
begin

definition *zs* ≡ *string-to-symbols x*

lemma *bit-symbols-zs*: *bit-symbols zs*
using *zs-def* **by** *simp*

abbreviation *n* ≡ *length x*

lemma *length-zs* [*simp*]: *length zs* = *n*
using *zs-def* **by** *simp*

definition *tps0* ≡ *snd* (*start-config* (*k1* + *k2*) *zs*)

definition *tps1a* :: *tape list* **where**
tps1a ≡ *SOME tps. tps* ::: *1* = *string-to-contents* (*f1 x*) ∧
transforms M1 (*snd* (*start-config k1* (*string-to-symbols x*))) (*T1 n*) *tps*

lemma *tps1a-aux*:
tps1a ::: *1* = *string-to-contents* (*f1 x*)
transforms M1 (*snd* (*start-config k1* (*string-to-symbols x*))) (*T1 n*) *tps1a*
using *tps1a-def someI-ex*[*OF computes-in-timeD*[*OF computes1*, *of x*]]
by *simp-all*

lemma *tps1a*:
tps1a ::: *1* = *string-to-contents* (*f1 x*)
transforms M1 (*snd* (*start-config k1 zs*)) (*T1 n*) *tps1a*
using *tps1a-aux zs-def* **by** *simp-all*

lemma *length-tps1a* [*simp*]: *length tps1a* = *k1*
using *tps1a*(*2*) *tm-M1 start-config-length execute-num-tapes transforms-def transits-def turing-machine-def*
by (*smt* (*verit*, *del-insts*) *Suc-1 add-pos-pos less-le-trans less-numeral-extra*(*1*) *plus-1-eq-Suc snd-conv*)

definition *tps1b* :: *tape list* **where**

```
  tps1b ≡ replicate k2 (⌊[]⌋, 0)

definition tps1 :: tape list where
  tps1 ≡ tps1a @ tps1b

lemma tps1-at-1: tps1 ! 1 = tps1a ! 1
  using tps1-def length-tps1a k-ge by (metis Suc-1 Suc-le-lessD nth-append)

lemma tps1-at-1': tps1 ::: 1 = string-to-contents (f1 x)
  using tps1-at-1 tps1a by simp

lemma tps1-pos-le: tps1 :#: 1 ≤ T1 n
proof -
  have execute M1 (start-config k1 zs) (T1 n) = (length M1, tps1a)
    using transforms-def transits-def tps1a(2)
    by (metis (no-types, lifting) execute-after-halting-ge fst-conv start-config-def snd-conv)
  moreover have execute M1 (start-config k1 zs) (T1 n) <#> 1 ≤ T1 n
    using head-pos-le-time[OF tm-M1, of 1] k-ge by fastforce
  ultimately show ?thesis
    using tps1-at-1 by simp
qed

lemma length-f1-x: length (f1 x) ≤ T1 n
proof -
  have execute M1 (start-config k1 zs) (T1 n) = (length M1, tps1a)
    using transforms-def transits-def tps1a(2)
    by (metis (no-types, lifting) execute-after-halting-ge fst-conv start-config-def snd-conv)
  moreover have (execute M1 (start-config k1 zs) (T1 n) <:> 1) i = □ if i > T1 n for i
    using blank-after-time[OF that - - tm-M1] k-ge(1) by simp
  ultimately have (tps1a ::: 1) i = □ if i > T1 n for i
    using that by simp
  then have (string-to-contents (f1 x)) i = □ if i > T1 n for i
    using that tps1a(1) by simp
  then have length (string-to-symbols (f1 x)) ≤ T1 n
    by (metis length-map order-refl verit-comp-simplify1(3) zero-neq-numeral zero-neq-one)
  then show ?thesis
    by simp
qed

lemma start-config-append:
  start-config (k1 + k2) zs = (0, snd (start-config k1 zs) @ tps1b)
proof
  have k1 > 0
    using tm-M1 turing-machine-def by simp
  show fst (start-config (k1 + k2) zs) = fst (0, snd (start-config k1 zs) @ tps1b)
    using start-config-def by simp
  show snd (start-config (k1 + k2) zs) = snd (0, snd (start-config k1 zs) @ tps1b)
    (is ?l = ?r)
  proof (rule nth-equalityI)
    have len: ||start-config k1 zs|| = k1
      using start-config-length by (simp add: ‹0 < k1›)
    show length ?l = length ?r
      using start-config-length tps1b-def tm-M1 turing-machine-def by simp
    show ?l ! j = ?r ! j if j < length ?l for j
    proof (cases j < k1)
      case True
      show ?thesis
      proof (cases j = 0)
        case True
        then show ?thesis
          using start-config-def ‹k1 > 0› by simp
      next
        case False
```

```
        then have 1: ?l ! j = (λi. if i = 0 then ▷ else □, 0)
          using start-config-def ‹k1 > 0› True by auto
        have ?r ! j = snd (start-config k1 zs) ! j
          using True len by (simp add: nth-append)
        then have ?r ! j = (λi. if i = 0 then ▷ else □, 0)
          using start-config4 ‹k1 > 0› False True by simp
        then show ?thesis
          using 1 by simp
      qed
    next
      case False
      then have j: j < k1 + k2 k1 ≤ j
        using that ‹0 < k1› add-gr-0 start-config-length by simp-all
      then have ?r ! j = (⌊[]⌋, 0)
        using tps1b-def by (simp add: False len nth-append)
      moreover have ?l ! j = (λi. if i = 0 then ▷ else □, 0)
        using start-config4 ‹k1 > 0› j by simp
      ultimately show ?thesis
        by auto
    qed
  qed
qed

lemma tm1 [transforms-intros]: transforms tm1 tps0 (T1 n) tps1
proof −
  let ?M = append-tapes k1 (k1 + length tps1b) M1
  have len: ||start-config k1 zs|| = k1
    using start-config-length[of k1 zs] tm-M1 turing-machine-def by simp
  have transforms ?M (snd (start-config k1 zs) @ tps1b) (T1 n) (tps1a @ tps1b)
    using transforms-append-tapes[OF tm-M1 len tps1a(2), of tps1b] .
  moreover have tps0 = snd (start-config k1 zs) @ tps1b
    unfolding tps0-def using start-config-append by simp
  ultimately have ∗: transforms ?M tps0 (T1 n) tps1
    using tps1-def by simp

  have symbols-lt G1 zs
    using bit-symbols-zs tm-M1 turing-machine-def by auto
  moreover have turing-machine (k1 + k2) G1 ?M
    using append-tapes-tm[OF tm-M1, of k1 + k2] by (simp add: tps1b-def)
  ultimately have transforms (enlarged G1 ?M) tps0 (T1 n) tps1
    using transforms-enlarged ∗ tps0-def by simp
  then show ?thesis
    using tm1-def tps1b-def by simp
qed

lemma clean-string-to-contents: clean-tape (string-to-contents xs, i)
  using clean-tape-def by simp

definition tps2 :: tape list where
  tps2 ≡ tps1 [1 := tps1 ! 1 |#=| 0]

lemma length-tps2 [simp]: length tps2 = k1 + k2
  using tps2-def tps1-def by (simp add: tps1b-def)

lemma tm2:
  assumes t = Suc (T1 n + tps1 :#: Suc 0)
  shows transforms tm2 tps0 t tps2
  unfolding tm2-def
proof (tform tps: assms tps2-def)
  show 1 < length tps1
    using tm-M1 turing-machine-def tps1-def by simp
  show clean-tape (tps1 ! 1)
      using tps1a(1) tps1-at-1 clean-tape-def by simp
```

qed

corollary *tm2′* [*transforms-intros*]:
  assumes *t* = *Suc* (*2* ∗ *T1 n*)
  shows *transforms tm2 tps0 t tps2*
  using *assms tm2 tps1-pos-le transforms-monotone* **by** *simp*

definition *tps3* :: *tape list* **where**
  tps3 ≡ *tps2* [*1* := *tps2* ! *1* |#=| (*Suc* (*length* (*f1 x*))), *k1* := *tps2* ! *1* |#=| (*Suc* (*length* (*f1 x*)))]

lemma *tm3*:
  assumes *t* = *Suc* (*Suc* (*2* ∗ *T1 n* + *Suc* (*length* (*f1 x*))))
  shows *transforms tm3 tps0 t tps3*
  unfolding *tm3-def*
proof (*tform tps*: *k-ge*)
  have *Suc 0* < *k1* + *k2 0* < *k2*
    using *k-ge* **by** *simp-all*
  then have ∗: *tps2* ! *1* = *tps1* ! *1* |#=| *0*
    using *tps2-def* **by** (*simp add*: *tps1-def tps1b-def*)
  let *?i* = *Suc* (*length* (*f1 x*))
  show *rneigh* (*tps2* ! *1*) {*0*} *?i*
    using ∗ *tps1-at-1 tps1a* **by** (*intro rneighI*) *auto*
  show *tps3* = *tps2*
    [*1* := *tps2* ! *1* |+| *Suc* (*length* (*f1 x*)),
     k1 := *implant* (*tps2* ! *1*) (*tps2* ! *k1*) (*Suc* (*length* (*f1 x*)))]
  proof −
    have *tps2* ! *1* |#=| (*Suc* (*length* (*f1 x*))) = *tps2* ! *1* |#=| *Suc* (*tps2* :#: *1* + *length* (*f1 x*))
      by (*metis* ∗ *One-nat-def add-Suc plus-1-eq-Suc snd-conv*)
    moreover have *tps2* ! *1* |#=| *?i* = *implant* (*tps2* ! *1*) (*tps2* ! *k1*) *?i*
    proof
      have *1*: *tps2* ! *1* = (*string-to-contents* (*f1 x*), *0*)
        using *tps1-at-1′* ∗ **by** *simp*
      have *tps1* ! *k1* = (⌊[]⌋, *0*)
        using *tps1-def tps1b-def* **by** (*simp add*: ‹*0* < *k2*› *nth-append*)
      then have *2*: *tps2* ! *k1* = (⌊[]⌋, *0*)
        using *tps2-def k-ge* **by** *simp*
      then show *snd* (*tps2* ! *1* |#=| *?i*) = *snd* (*implant* (*tps2* ! *1*) (*tps2* ! *k1*) *?i*)
        using *implant* **by** *simp*
      have *fst* (*implant* (*tps2* ! *1*) (*tps2* ! *k1*) *?i*) *i* = *fst* (*tps2* ! *1* |#=| *?i*) *i* **for** *i*
        using *1 2 implant* **by** *simp*
      then show *fst* (*tps2* ! *1* |#=| *?i*) = *fst* (*implant* (*tps2* ! *1*) (*tps2* ! *k1*) *?i*)
        by *auto*
    qed
    ultimately show *?thesis*
      using *tps3-def* **by** *simp*
  qed
  show *t* = *Suc* (*2* ∗ *T1 n*) + *Suc* (*Suc* (*length* (*f1 x*)))
    using *assms* **by** *simp*
qed

definition *tps3′* ≡ *tps1a*
  [*1* := (*string-to-contents* (*f1 x*), *Suc* (*length* (*f1 x*)))] @
   ((*string-to-contents* (*f1 x*), *Suc* (*length* (*f1 x*))) #
    replicate (*k2* − *1*) (⌊[]⌋, *0*))

lemma *tps3′*: *tps3* = *tps3′*
proof (*rule nth-equalityI*)
  have *length tps3* = *k1* + *k2*
    using *tps3-def* **by** *simp*
  moreover have *length tps3′* = *k1* + *k2*
    using *k-ge*(*2*) *tps3′-def* **by** *simp*
  ultimately show *length tps3* = *length tps3′*
    by *simp*

```
show tps3 ! j = tps3' ! j if j < length tps3 for j
proof (cases j < k1)
  case True
  then have rhs: tps3' ! j = (tps1a [1 := (string-to-contents (f1 x), Suc (length (f1 x)))]) ! j
    by (simp add: tps3'-def nth-append)
  show ?thesis
  proof (cases j = 1)
    case True
    then have tps3 ! j = tps2 ! 1 |#=| (Suc (length (f1 x)))
      using tps3-def Suc-1 Suc-n-not-le-n ‹length tps3 = k1 + k2› k-ge(1)
        length-tps2 nth-list-update-eq nth-list-update-neq that
      by auto
    then have tps3 ! j = tps1 ! 1 |#=| (Suc (length (f1 x)))
      using tps2-def True ‹length tps3 = k1 + k2› length-tps2 that by auto
    then have tps3 ! j = (string-to-contents (f1 x), Suc (length (f1 x)))
      using tps1-at-1 tps1a(1) by simp
    then show ?thesis
      using rhs True k-ge(1) by auto
  next
    case False
    then have tps3 ! j = tps2 ! j
      using tps3-def True by simp
    then have tps3 ! j = tps1 ! j
      using tps2-def False by simp
    then have tps3 ! j = tps1a ! j
      using length-tps1a tps1-def False True by (simp add: nth-append)
    moreover have tps3' ! j = tps1a ! j
      using False rhs by simp
    ultimately show ?thesis
      by simp
  qed
next
  case j-ge: False
  show ?thesis
  proof (cases j = k1)
    case True
    then have tps3 ! j = tps2 ! 1 |#=| (Suc (length (f1 x)))
      using tps3-def that by simp
    then have tps3 ! j = tps1 ! 1 |#=| (Suc (length (f1 x)))
      using tps2-def ‹length tps3 = k1 + k2› length-tps2 Suc-1 Suc-le-lessD tm1-tm turing-machine-def
      by simp
    then have tps3 ! j = (string-to-contents (f1 x), Suc (length (f1 x)))
      using tps1-at-1 tps1a(1) by simp
    moreover have tps3' ! j = (string-to-contents (f1 x), Suc (length (f1 x)))
      using True tps3'-def
      by (metis (no-types, lifting) length-list-update length-tps1a nth-append-length)
    ultimately show ?thesis
      by simp
  next
    case False
    then have j > k1
      using j-ge by simp
    then have tps3' ! j = ((string-to-contents (f1 x), Suc (length (f1 x))) #
        replicate (k2 − 1) (⌊[]⌋, 0)) ! (j − k1)
      by (simp add: tps3'-def nth-append)
    moreover have j − k1 < k2
      by (metis ‹k1 < j› ‹length tps3 = k1 + k2› add.commute less-diff-conv2 less-imp-le that)
    ultimately have *: tps3' ! j = (⌊[]⌋, 0)
      by (metis (no-types, lifting) Suc-leI ‹k1 < j› add-leD1 le-add-diff-inverse2 less-diff-conv2
        nth-Cons-pos nth-replicate plus-1-eq-Suc zero-less-diff)
    have tps3 ! j = tps2 ! j
      using tps3-def ‹k1 < j› k-ge(1) by simp
    then have tps3 ! j = tps1 ! j
```

    using *tps2-def* ‹*k1* < *j*› *k-ge*(*1*) **by** *simp*
   then have *tps3* ! *j* = *tps1b* ! (*j* − *k1*)
    using *tps1-def* **by** (*simp add*: *j-ge nth-append*)
   then have *tps3* ! *j* = (⌊[]⌋, *0*)
    using *tps1b-def* **by** (*simp add*: ‹*j* − *k1* < *k2*›)
   then show *?thesis*
    using ∗ **by** *simp*
  qed
 qed
qed

lemma *tm3′* [*transforms-intros*]:
 assumes *t* = *Suc* (*Suc* (*Suc* (*3* ∗ *T1 n*)))
 shows *transforms tm3 tps0 t tps3′*
proof −
 have *transforms tm3 tps0* (*Suc* (*Suc* (*2* ∗ *T1 n* + *Suc* (*length* (*f1 x*))))) *tps3*
  using *tm3* **by** *simp*
 moreover have *t* ≥ *Suc* (*Suc* (*2* ∗ *T1 n* + *Suc* (*length* (*f1 x*))))
  using *assms length-f1-x* **by** *simp*
 ultimately show *?thesis*
  using *tps3′ transforms-monotone* **by** *auto*
qed

definition *tps4* ≡
 tps1a [*1* := (⌊[]⌋, *1*)] @
 ((*string-to-contents* (*f1 x*), *Suc* (*length* (*f1 x*))) #
  replicate (*k2* − *1*) (⌊[]⌋, *0*))

lemma *tm4*:
 assumes *t* = *9* + (*3* ∗ *T1 n* + (*Suc* (*3* ∗ *length* (*string-to-symbols* (*f1 x*)))))
 shows *transforms tm4 tps0 t tps4*
 unfolding *tm4-def*
proof (*tform*)
 show *1* < *length tps3′*
  using *tps3′-def* **using** *tm1-tm turing-machine-def* **by** *auto*
 let *?zs* = *string-to-symbols* (*f1 x*)
 show *proper-symbols ?zs*
  by *simp*
 show *tps4* = *tps3′*[*1* := (⌊[]⌋, *1*)]
  using *tps4-def tps3′-def k-ge*(*1*) *length-tps1a* **by** (*simp add*: *list-update-append1*)
 show *tps3′* ::: *1* = ⌊*string-to-symbols* (*f1 x*)⌋
 proof −
  have *tps3′* ! *1* = (*string-to-contents* (*f1 x*), *Suc* (*length* (*f1 x*)))
   using *tps3′-def k-ge*(*1*) *length-tps1a* **by** (*simp add*: *nth-append*)
  then show *?thesis*
   by *auto*
 qed
 have *tps3′* :#: *1* = *Suc* (*length* (*f1 x*))
  using *tps3′-def k-ge*(*1*) *length-tps1a* **by** (*simp add*: *nth-append*)
 then show *t* = *Suc* (*Suc* (*Suc* (*3* ∗ *T1 n*))) +
   (*tps3′* :#: *1* + *2* ∗ *length* (*string-to-symbols* (*f1 x*)) + *6*)
  using *assms* **by** *simp*
qed

lemma *tm4′* [*transforms-intros*]:
 assumes *t* = *10* + (*6* ∗ *T1 n*)
 shows *transforms tm4 tps0 t tps4*
proof (*rule transforms-monotone*[*OF tm4*], *simp*)
 show *10* + (*3* ∗ *T1 n* + *3* ∗ *length* (*f1 x*)) ≤ *t*
  using *length-f1-x assms* **by** *simp*
qed

definition *tps5* ≡

tps1a [*1* := (⌊[]⌋, *1*)] @
((*string-to-contents* (*f1 x*), *0*) #
 replicate (*k2* − *1*) (⌊[]⌋, *0*))

lemma *tm5*:
  assumes *t* = *11* + (*6* ∗ *T1 n* + *tps4* :#: *k1*)
  shows *transforms tm5 tps0 t tps5*
  unfolding *tm5-def*
proof (*tform time*: *assms*)
  show *k1* < *length tps4*
    using *tps4-def length-tps1a* **by** *simp*
  show *tps5* = *tps4*[*k1* := *tps4* ! *k1* |#=| *0*]
    using *tps4-def tps5-def length-tps1a*
    by (*metis* (*no-types*, *lifting*) *fst-conv length-list-update list-update-length nth-append-length*)
  show *clean-tape* (*tps4* ! *k1*)
    using *tps4-def length-tps1a clean-tape-def*
    by (*smt* (*verit*) *Suc-eq-plus1 add.commute add-cancel-right-right*
      fst-conv length-list-update nat.distinct(*1*) *nat-1-add-1 nth-append-length numeral-3-eq-3*)
qed

abbreviation *ys* ≡ *string-to-symbols* (*f1 x*)

abbreviation *m* ≡ *length* (*f1 x*)

definition *tps5′* ≡
  tps1a [*1* := (⌊[]⌋, *1*)] @
  snd (*start-config k2 ys*)

lemma *tps5′*: *tps5* = *tps5′*
  using *tps5-def tps5′-def start-config-def* **by** *auto*

lemma *tm5′* [*transforms-intros*]:
  assumes *t* = *12* + *7* ∗ *T1 n*
  shows *transforms tm5 tps0 t tps5′*
proof −
  have *tps4* :#: *k1* = *Suc* (*length* (*f1 x*))
    using *tps4-def*
    by (*metis* (*no-types*, *lifting*) *length-list-update length-tps1a nth-append-length snd-conv*)
  then have *tps4* :#: *k1* ≤ *Suc* (*T1 n*)
    using *length-f1-x* **by** *simp*
  then have *t* ≥ *11* + (*6* ∗ *T1 n* + *tps4* :#: *k1*)
    using *assms* **by** *simp*
  then show *?thesis*
    using *tm5 transforms-monotone tps5′* **by** *simp*
qed

definition *tps6b* :: *tape list* **where**
  tps6b ≡ *SOME tps*. *tps* ::: *1* = *string-to-contents* (*f2* (*f1 x*)) ∧
      transforms M2 (*snd* (*start-config k2 ys*)) (*T2 m*) *tps*

lemma *tps6b*:
  tps6b ::: *1* = *string-to-contents* (*f2* (*f1 x*))
  transforms M2 (*snd* (*start-config k2 ys*)) (*T2 m*) *tps6b*
  using *tps6b-def someI-ex*[*OF computes-in-timeD*[*OF computes2*, *of f1 x*]]
  by *simp-all*

lemma *tps6b-pos-le*: *tps6b* :#: *1* ≤ *T2 m*
proof −
  have *execute M2* (*start-config k2 ys*) (*T2 m*) = (*length M2*, *tps6b*)
    using *transforms-def transits-def tps6b*(*2*)
    by (*metis* (*no-types*, *lifting*) *execute-after-halting-ge fst-conv start-config-def snd-conv*)
  moreover have *execute M2* (*start-config k2 ys*) (*T2 m*) <#> *1* ≤ *T2 m*
    using *head-pos-le-time*[*OF tm-M2*, *of 1*] *k-ge* **by** *simp*

ultimately show *?thesis*
by *simp*
qed

lemma *length-tps6b*: *length tps6b* = *k2*
using *tm-M2 execute-num-tapes k-ge*(*2*) *tps5′ tps5′-def tps5-def tps6b*(*2*) *transforms-def transits-def*
by (*smt* (*verit*, *ccfv-threshold*) *One-nat-def Suc-diff-Suc length-Cons length-replicate less-le-trans minus-nat.diff-0 numeral-2-eq-2 prod.sel*(*2*) *same-append-eq zero-less-Suc*)

lemma *length-f2-f1-x*: *length* (*f2* (*f1 x*)) ≤ *T2 m*
proof −
have *execute M2* (*start-config k2 ys*) (*T2 m*) = (*length M2*, *tps6b*)
using *transforms-def transits-def tps6b*(*2*)
by (*metis* (*no-types*, *lifting*) *execute-after-halting-ge fst-conv start-config-def snd-conv*)
moreover have (*execute M2* (*start-config k2 ys*) (*T2 m*) <:> *1*) *i* = *0* **if** *i* > *T2 m* **for** *i*
using *blank-after-time*[*OF that - - tm-M2*] *k-ge*(*2*) **by** *simp*
ultimately have (*tps6b* ::: *1*) *i* = □ **if** *i* > *T2 m* **for** *i*
using *that* **by** *simp*
then have (*string-to-contents* (*f2* (*f1 x*))) *i* = □ **if** *i* > *T2 m* **for** *i*
using *that tps6b*(*1*) **by** *simp*
then have *length* (*string-to-symbols* (*f2* (*f1 x*))) ≤ *T2 m*
by (*metis length-map order-refl verit-comp-simplify1*(*3*) *zero-neq-numeral zero-neq-one*)
then show *?thesis*
by *simp*
qed

lemma *enlarged-M2*: *transforms* (*enlarged G2 M2*) (*snd* (*start-config k2 ys*)) (*T2 m*) *tps6b*
proof −
have *symbols-lt G2* (*string-to-symbols* (*f1 x*))
using *tm-M2 turing-machine-def* **by** *simp*
then show *?thesis*
using *transforms-enlarged*[*OF tm-M2 - tps6b*(*2*)] **by** *simp*
qed

lemma *enlarged-M2-tm*: *turing-machine k2 G* (*enlarged G2 M2*)
using *turing-machine-enlarged tm-M2 G2* **by** *simp*

definition *tps6* ≡ *tps1a*[*1* := (⌊[]⌋, *1*)] @ *tps6b*

lemma *tm56* [*transforms-intros*]: *transforms tm56 tps5′* (*T2 m*) *tps6*
using *transforms-prepend-tapes*[*OF enlarged-M2-tm - - enlarged-M2*, *of tps1a* [*1* := (⌊[]⌋, *1*)] *k1*]
tps5′-def tps6-def tm56-def start-config-length k-ge(*2*)
by *auto*

lemma *tps6-at-Suc-k1*: *tps6* ::: (*k1* + *1*) = *string-to-contents* (*f2* (*f1 x*))
and *tps6-pos-le*: *tps6* :#: (*k1* + *1*) ≤ *T2 m*
proof −
have *tps6* ! (*k1* + *1*) = *tps6b* ! *1*
using *tps6-def length-tps1a length-tps6b* **by** (*simp add*: *nth-append*)
then show
tps6 ::: (*k1* + *1*) = *string-to-contents* (*f2* (*f1 x*))
tps6 :#: (*k1* + *1*) ≤ *T2 m*
using *tps6b*(*1*) *tps6b-pos-le* **by** *simp-all*
qed

lemma *tm6* [*transforms-intros*]:
assumes *t* = *12* + *7* ∗ *T1 n* + *T2 m*
shows *transforms tm6 tps0 t tps6*
unfolding *tm6-def* **by** (*tform tps*: *assms*)

definition *tps7* ≡
tps1a[*1* := (⌊[]⌋, *1*)] @
tps6b[*1* := (*string-to-contents* (*f2* (*f1 x*)), *1*)]

lemma *tps7-at-Suc-k1*: *tps7* ! (*k1* + *1*) = (*string-to-contents* (*f2* (*f1 x*)), *1*)
  using *tps7-def k-ge*(*2*) *length-tps1a length-tps6b*
  by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-le-lessD add.commute diff-add-inverse*
    length-list-update not-add-less2 nth-append nth-list-update-eq numeral-2-eq-2)

lemma *tm7*:
  assumes $t = 14 + (7 * T1\ n + (T2\ m + tps6 :\#: Suc\ k1))$
  shows *transforms tm7 tps0 t tps7*
  unfolding *tm7-def*
proof (*tform time*: *assms*)
  show *k1* + *1* < *length tps6*
    using *tps6-def k-ge*(*2*) *length-tps1a length-tps6b* **by** *simp*
  show *clean-tape* (*tps6* ! (*k1* + *1*))
    using *tps6-at-Suc-k1 clean-tape-def* **by** *simp*
  show *tps7* = *tps6*[*k1* + *1* := *tps6* ! (*k1* + *1*) |#=| *1*]
  proof −
    have *tps6* ! (*k1* + *1*) |#=| *1* = (*string-to-contents* (*f2* (*f1 x*)), *1*)
      using *tps6-at-Suc-k1* **by** *simp*
    then show *?thesis*
      using *tps6-def tps7-def length-tps1a length-tps6b k-ge tps7-at-Suc-k1*
      by (*metis* (*no-types*, *lifting*) *add.commute diff-add-inverse length-list-update*
        list-update-append not-add-less2 plus-1-eq-Suc)
  qed
qed

corollary *tm7′* [*transforms-intros*]:
  assumes $t = 14 + 7 * T1\ n + 2 * T2\ m$
  shows *transforms tm7 tps0 t tps7*
proof (*rule transforms-monotone*[*OF tm7*], *simp*)
  show $14 + (7 * T1\ n + (T2\ (length\ (f1\ x)) + tps6 :\#: Suc\ k1)) \le t$
    using *assms tps6-pos-le* **by** *simp*
qed

definition *tps8* ≡
  tps1a[*1* := (*string-to-contents* (*f2* (*f1 x*)), *Suc* (*length* (*f2* (*f1 x*))))] @
  tps6b[*1* := (*string-to-contents* (*f2* (*f1 x*)), *Suc* (*length* (*f2* (*f1 x*))))]

lemma *tps8-at-1*: *tps8* ::: *1* = *string-to-contents* (*f2* (*f1 x*))
  using *tps8-def length-tps1a k-ge*(*1*)
  by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-le-lessD length-list-update nth-append*
    nth-list-update-eq numeral-2-eq-2 prod.sel(*1*))

lemma *tm8*:
  assumes $t = 15 + 7 * T1\ n + 2 * T2\ m + length\ (f2\ (f1\ x))$
  shows *transforms tm8 tps0 t tps8*
  unfolding *tm8-def*
proof (*tform tps*: *assms*)
  show *k1* + *1* < *length tps7*
    using *tps7-def length-tps1a length-tps6b k-ge*(*2*) **by** *simp*
  show *1* < *length tps7*
    using *tps7-def length-tps6b k-ge*(*2*) **by** *simp*
  let *?n* = *length* (*f2* (*f1 x*))
  show *rneigh* (*tps7* ! (*k1* + *1*)) {□} *?n*
  proof (*rule rneighI*)
   show (*tps7* ::: (*k1* + *1*)) (*tps7* :#: (*k1* + *1*) + *?n*) ∈ {□}
     using *tps7-at-Suc-k1* **by** *simp*
   show $\bigwedge n'.\ n' < ?n \Longrightarrow (tps7 ::: (k1 + 1))\ (tps7 :\#: (k1 + 1) + n') \notin \{\Box\}$
     using *tps7-at-Suc-k1* **by** *simp*
  qed
  show *tps8* = *tps7*
    [*k1* + *1* := *tps7* ! (*k1* + *1*) |+| *?n*,
     1 := *implant* (*tps7* ! (*k1* + *1*)) (*tps7* ! *1*) *?n*]

```
  (is tps8 = ?tps)
proof (rule nth-equalityI)
  show length tps8 = length ?tps
    using tps8-def tps7-def by simp
  have len: length tps8 = k1 + k2
    using tps8-def length-tps6b by simp
  show tps8 ! j = ?tps ! j if j < length tps8 for j
  proof (cases j < k1)
    case j-less: True
    then have lhs: tps8 ! j = tps1a[1 := (string-to-contents (f2 (f1 x)), Suc (length (f2 (f1 x))))] ! j
      using tps8-def length-tps1a length-tps6b k-ge by (simp add: nth-append)
    show ?thesis
    proof (cases j = 1)
      case True
      then have 1: ?tps ! j = implant (tps7 ! (k1 + 1)) (tps7 ! 1) ?n
        using ‹1 < length tps7› by simp
      have 2: tps8 ! j = (string-to-contents (f2 (f1 x)), Suc (length (f2 (f1 x))))
        using lhs True j-less by simp
      have 3: tps7 ! 1 = (⌊[]⌋, 1)
        using tps7-def length-tps1a
        by (metis (no-types, lifting) True j-less length-list-update nth-append nth-list-update-eq)
      have implant (string-to-contents (f2 (f1 x)), 1) (⌊[]⌋, 1) ?n =
          (string-to-contents (f2 (f1 x)), Suc ?n)
        using implant contents-def by auto
      then show ?thesis
        using 1 2 3 tps7-at-Suc-k1 by simp
    next
      case False
      then have ?tps ! j = tps7 ! j
        by (metis One-nat-def Suc-eq-plus1 add.commute j-less not-add-less2 nth-list-update-neq)
      then have ?tps ! j = tps1a ! j
        using False j-less tps7-def length-tps1a
        by (metis (no-types, lifting) length-list-update nth-append nth-list-update-neq)
      moreover have tps8 ! j = tps1a ! j
        using False j-less tps8-def lhs by simp
      ultimately show ?thesis
        by simp
    qed
  next
    case j-ge: False
    then have lhs: tps8 ! j = tps6b[1 := (string-to-contents (f2 (f1 x)), Suc (length (f2 (f1 x))))] ! (j − k1)
      using tps8-def length-tps1a length-tps6b k-ge by (simp add: nth-append)
    show ?thesis
    proof (cases j = Suc k1)
      case True
      then have tps8 ! j = (string-to-contents (f2 (f1 x)), Suc (length (f2 (f1 x))))
        using lhs len that length-tps6b by simp
      moreover have ?tps ! j = tps7 ! Suc k1 |+| ?n
        using True ‹k1 + 1 < length tps7› k-ge(1) by simp
      ultimately show ?thesis
        using tps7-at-Suc-k1 True by simp
    next
      case False
      then have tps8 ! j = tps6b ! (j − k1)
        using lhs by simp
      moreover have ?tps ! j = tps7 ! j
        using False j-ge that k-ge(1) by simp
      ultimately show ?thesis
        using tps7-def j-ge False length-tps1a
        by (metis (no-types, lifting) add.commute add-diff-inverse-nat length-list-update
          nth-append nth-list-update-neq plus-1-eq-Suc)
    qed
  qed
```

```
  qed
qed

lemma tm8':
  assumes t = 15 + 7 * T1 n + 3 * T2 m
  shows transforms tm8 tps0 t tps8
proof (rule transforms-monotone[OF tm8], simp)
  show 15 + 7 * T1 n + 2 * T2 m + length (f2 (f1 x)) ≤ t
    using length-f2-f1-x assms by simp
qed

lemma tm8'-mono:
  assumes mono T2
    and t = 15 + 7 * T1 n + 3 * T2 (T1 n)
  shows transforms tm8 tps0 t tps8
proof (rule transforms-monotone[OF tm8'], simp)
  have T2 (T1 n) ≥ T2 m
    using assms(1) length-f1-x monoE by auto
  then show 15 + 7 * T1 n + 3 * T2 m ≤ t
    using assms(2) by simp
qed

end

lemma computes-composed-mono:
  assumes mono T2 and T = (λn. 15 + 7 * T1 n + 3 * T2 (T1 n))
  shows computes-in-time (k1 + k2) tm8 (f2 ∘ f1) T
proof
  fix x
  have tps8 x ::: 1 = string-to-contents (f2 (f1 x))
    using tps8-at-1 by simp
  moreover have transforms tm8 (snd (start-config (k1 + k2) (string-to-symbols x))) (T (length x)) (tps8 x)
    using tm8'-mono assms tps0-def zs-def by simp
  ultimately show ∃tps.
      tps ::: 1 = string-to-contents ((f2 ∘ f1) x) ∧
      transforms tm8 (snd (start-config (k1 + k2) (string-to-symbols x))) (T (length x)) tps
    by force
qed

end

lemma computes-composed-poly:
  assumes tm-M1: turing-machine k1 G1 M1
    and tm-M2: turing-machine k2 G2 M2
    and computes1: computes-in-time k1 M1 f1 T1
    and computes2: computes-in-time k2 M2 f2 T2
  assumes big-oh-poly T1 and big-oh-poly T2
  shows ∃T k G M. big-oh-poly T ∧ turing-machine k G M ∧ computes-in-time k M (f2 ∘ f1) T
proof -
  obtain d1 :: nat where big-oh T1 (λn. n ^ d1)
    using assms(5) big-oh-poly-def by auto
  obtain b c d2 :: nat where cm: d2 > 0 ∀n. T2 n ≤ b + c * n ^ d2
    using big-oh-poly-offset[OF assms(6)] by auto
  let ?U = λn. b + c * n ^ d2
  have U: T2 n ≤ ?U n for n
    using cm by simp
  have mono ?U
    by standard (simp add: cm(1))
  have computesU: computes-in-time k2 M2 f2 ?U
    using computes-in-time-mono[OF computes2 U] .
  interpret compo: compose k1 k2 G1 G2 M1 M2 T1 ?U f1 f2
    using assms computesU compose.intro by simp
  let ?M = compo.tm8
```

```
  let ?T = (λn. 15 + 7 * T1 n + 3 * (b + c * T1 n ^ d2))
  have computes-in-time (k1 + k2) ?M (f2 ∘ f1) ?T
    using compo.computes-composed-mono[OF ‹mono ?U›, of ?T] by simp
  moreover have big-oh-poly ?T
  proof −
    have big-oh-poly (λn. n ^ d2)
      using big-oh-poly-poly by simp
    moreover have (λn. T1 n ^ d2) = (λn. n ^ d2) ∘ T1
      by auto
    ultimately have big-oh-poly (λn. T1 n ^ d2)
      using big-oh-poly-composition[OF assms(5)] by auto
    then have big-oh-poly (λn. 3 * (b + c * T1 n ^ d2))
      using big-oh-poly-const big-oh-poly-prod big-oh-poly-sum by simp
    then show ?thesis
      using assms(5) big-oh-poly-const big-oh-poly-prod big-oh-poly-sum by simp
  qed
  moreover have turing-machine (k1 + k2) compo.G ?M
    using compo.tm8-tm .
  ultimately show ?thesis
    by auto
qed

end
```

2.7 Arithmetic

```
theory Arithmetic
  imports Memorizing
begin
```

In this section we define a representation of natural numbers and some reusable Turing machines for elementary arithmetical operations. All Turing machines implementing the operations assume that the tape heads on the tapes containing the operands and the result(s) contain one natural number each. In programming language terms we could say that such a tape corresponds to a variable of type *nat*. Furthermore, initially the tape heads are on cell number 1, that is, one to the right of the start symbol. The Turing machines will halt with the tape heads in that position as well. In that way operations can be concatenated seamlessly.

2.7.1 Binary numbers

We represent binary numbers as sequences of the symbols **0** and **1**. Slightly unusually the least significant bit will be on the left. While every sequence over these symbols represents a natural number, the representation is not unique due to leading (or rather, trailing) zeros. The *canonical* representation is unique and has no trailing zeros, not even for the number zero, which is thus represented by the empty symbol sequence. As a side effect empty tapes can be thought of as being initialized with zero.

Naturally the binary digits 0 and 1 are represented by the symbols **0** and **1**, respectively. For example, the decimal number 14, conventionally written 1100_2 in binary, is represented by the symbol sequence **0011**. The next two functions map between symbols and binary digits:

```
abbreviation (input) tosym :: nat ⇒ symbol where
  tosym z ≡ z + 2

abbreviation todigit :: symbol ⇒ nat where
  todigit z ≡ if z = 1 then 1 else 0
```

The numerical value of a symbol sequence:

```
definition num :: symbol list ⇒ nat where
  num xs ≡ ∑i←[0..<length xs]. todigit (xs ! i) * 2 ^ i
```

The *i*-th digit of a symbol sequence, where digits out of bounds are considered trailing zeros:

```
definition digit :: symbol list ⇒ nat ⇒ nat where
```

digit xs i ≡ *if i* < *length xs then xs* ! *i else 0*

Some properties of *num*:

lemma *num-ge-pow*:
  assumes *i* < *length xs* **and** *xs* ! *i* = **1**
  shows *num xs* ≥ *2* ^ *i*
proof −
  let *?ys* = *map* (λ*i*. *todigit* (*xs* ! *i*) ∗ *2* ^ *i*) [*0*..<*length xs*]
  have *?ys* ! *i* = *2* ^ *i*
    using *assms* **by** *simp*
  moreover have *i* < *length ?ys*
    using *assms*(*1*) **by** *simp*
  ultimately show *num xs* ≥ *2* ^ *i*
    unfolding *num-def* **using** *elem-le-sum-list* **by** (*metis* (*no-types*, *lifting*))
qed

lemma *num-trailing-zero*:
  assumes *todigit z* = *0*
  shows *num xs* = *num* (*xs* @ [*z*])
proof −
  let *?xs* = *xs* @ [*z*]
  let *?ys* = *map* (λ*i*. *todigit* (*?xs* ! *i*) ∗ *2* ^ *i*) [*0*..<*length ?xs*]
  have ∗: *?ys* = *map* (λ*i*. *todigit* (*xs* ! *i*) ∗ *2* ^ *i*) [*0*..<*length xs*] @ [*0*]
    using *assms* **by** (*simp add*: *nth-append*)
  have *num ?xs* = *sum-list ?ys*
    using *num-def* **by** *simp*
  then have *num ?xs* = *sum-list* (*map* (λ*i*. *todigit* (*xs* ! *i*) ∗ *2* ^ *i*) [*0*..<*length xs*] @ [*0*])
    using ∗ **by** *metis*
  then have *num ?xs* = *sum-list* (*map* (λ*i*. *todigit* (*xs* ! *i*) ∗ *2* ^ *i*) [*0*..<*length xs*])
    by *simp*
  then show *?thesis*
    using *num-def* **by** *simp*
qed

lemma *num-Cons*: *num* (*x* # *xs*) = *todigit x* + *2* ∗ *num xs*
proof −
  have [*0*..<*length* (*x* # *xs*)] = [*0*..<*1*] @ [*1*..<*length* (*x* # *xs*)]
    by (*metis length-Cons less-imp-le-nat plus-1-eq-Suc upt-add-eq-append zero-less-one*)
  then have *1*: (*map* (λ*i*. *todigit* ((*x* # *xs*) ! *i*) ∗ *2* ^ *i*) [*0*..<*length* (*x* # *xs*)]) =
    (*map* (λ*i*. *todigit* ((*x* # *xs*) ! *i*) ∗ *2* ^ *i*) [*0*..<*1*]) @
    (*map* (λ*i*. *todigit* ((*x* # *xs*) ! *i*) ∗ *2* ^ *i*) [*1*..<*length* (*x* # *xs*)])
    by *simp*

  have *map* (λ*i*. *f i*) [*1*..<*Suc m*] = *map* (λ*i*. *f* (*Suc i*)) [*0*..<*m*] **for** *f* :: *nat* ⇒ *nat* **and** *m*
  proof (*rule nth-equalityI*)
    show *length* (*map f* [*1*..<*Suc m*]) = *length* (*map* (λ*i*. *f* (*Suc i*)) [*0*..<*m*])
      by *simp*
    then show ⋀*i*. *i* < *length* (*map f* [*1*..<*Suc m*]) ⟹
        map f [*1*..<*Suc m*] ! *i* = *map* (λ*i*. *f* (*Suc i*)) [*0*..<*m*] ! *i*
      by (*metis add.left-neutral length-map length-upt nth-map-upt plus-1-eq-Suc*)
  qed
  then have *2*: (∑*i*←[*1*..<*Suc m*]. *f i*) = (∑*i*←[*0*..<*m*]. *f* (*Suc i*))
      for *f* :: *nat* ⇒ *nat* **and** *m*
    by *simp*

  have *num* (*x* # *xs*) = (∑*i*←[*0*..<*length* (*x* # *xs*)]. *todigit* ((*x* # *xs*) ! *i*) ∗ *2* ^ *i*)
    using *num-def* **by** *simp*
  also have ... = (∑*i*←[*0*..<*1*]. (*todigit* ((*x* # *xs*) ! *i*) ∗ *2* ^ *i*)) +
      (∑*i*←[*1*..<*length* (*x* # *xs*)]. *todigit* ((*x* # *xs*) ! *i*) ∗ *2* ^ *i*)
    using *1* **by** *simp*
  also have ... = *todigit x* + (∑*i*←[*1*..<*length* (*x* # *xs*)]. *todigit* ((*x* # *xs*) ! *i*) ∗ *2* ^ *i*)
    by *simp*
  also have ... = *todigit x* + (∑*i*←[*0*..<*length* (*x* # *xs*) − *1*]. *todigit* ((*x* # *xs*) ! (*Suc i*)) ∗ *2* ^(*Suc i*))

```
    using 2 by simp
  also have ... = todigit x + (∑ i←[0..<length xs]. todigit (xs ! i) ∗ 2 ^ (Suc i))
    by simp
  also have ... = todigit x + (∑ i←[0..<length xs]. todigit (xs ! i) ∗ (2 ∗ 2 ^ i))
    by simp
  also have ... = todigit x + (∑ i←[0..<length xs]. (todigit (xs ! i) ∗ 2 ∗ 2 ^ i))
    by (simp add: mult.assoc)
  also have ... = todigit x + (∑ i←[0..<length xs]. (2 ∗ (todigit (xs ! i) ∗ 2 ^ i)))
    by (metis (mono-tags, opaque-lifting) ab-semigroup-mult-class.mult-ac(1) mult.commute)
  also have ... = todigit x + 2 ∗ (∑ i←[0..<length xs]. (todigit (xs ! i) ∗ 2 ^ i))
    using sum-list-const-mult by fastforce
  also have ... = todigit x + 2 ∗ num xs
    using num-def by simp
  finally show ?thesis .
qed

lemma num-append: num (xs @ ys) = num xs + 2 ^ length xs ∗ num ys
proof (induction length xs arbitrary: xs)
  case 0
  then show ?case
    using num-def by simp
next
  case (Suc n)
  then have xs: xs = hd xs # tl xs
    by (metis hd-Cons-tl list.size(3) nat.simps(3))
  then have xs @ ys = hd xs # (tl xs @ ys)
    by simp
  then have num (xs @ ys) = todigit (hd xs) + 2 ∗ num (tl xs @ ys)
    using num-Cons by presburger
  also have ... = todigit (hd xs) + 2 ∗ (num (tl xs) + 2 ^ length (tl xs) ∗ num ys)
    using Suc by simp
  also have ... = todigit (hd xs) + 2 ∗ num (tl xs) + 2 ^ Suc (length (tl xs)) ∗ num ys
    by simp
  also have ... = num xs + 2 ^ Suc (length (tl xs)) ∗ num ys
    using num-Cons xs by metis
  also have ... = num xs + 2 ^ length xs ∗ num ys
    using xs by (metis length-Cons)
  finally show ?case .
qed

lemma num-drop: num (drop t zs) = todigit (digit zs t) + 2 ∗ num (drop (Suc t) zs)
proof (cases t < length zs)
  case True
  then have drop t zs = zs ! t # drop (Suc t) zs
    by (simp add: Cons-nth-drop-Suc)
  then have num (drop t zs) = todigit (zs ! t) + 2 ∗ num (drop (Suc t) zs)
    using num-Cons by simp
  then show ?thesis
    using digit-def True by simp
next
  case False
  then show ?thesis
    using digit-def num-def by simp
qed

lemma num-take-Suc: num (take (Suc t) zs) = num (take t zs) + 2 ^ t ∗ todigit (digit zs t)
proof (cases t < length zs)
  case True
  let ?zs = take (Suc t) zs
  have 1: ?zs ! i = zs ! i if i < Suc t for i
    using that by simp
  have 2: take t zs ! i = zs ! i if i < t for i
    using that by simp
```

have *num ?zs* = ($\sum$ *i*←[*0*..<*length ?zs*]. *todigit* (*?zs* ! *i*) ∗ *2* ^ *i*)
using *num-def* **by** *simp*
also have ... = ($\sum$ *i*←[*0*..<*Suc t*]. *todigit* (*?zs* ! *i*) ∗ *2* ^ *i*)
by (*simp add*: *Suc-leI True min-absorb2*)
also have ... = ($\sum$ *i*←[*0*..<*Suc t*]. *todigit* (*zs* ! *i*) ∗ *2* ^ *i*)
using *1* **by** (*smt* (*verit*, *best*) *atLeastLessThan-iff map-eq-conv set-upt*)
also have ... = ($\sum$ *i*←[*0*..<*t*]. *todigit* (*zs* ! *i*) ∗ *2* ^ *i*) + *todigit* (*zs* ! *t*) ∗ *2* ^ *t*
by *simp*
also have ... = ($\sum$ *i*←[*0*..<*t*]. *todigit* (*take t zs* ! *i*) ∗ *2* ^ *i*) + *todigit* (*zs* ! *t*) ∗ *2* ^ *t*
using *2* **by** (*metis* (*no-types*, *lifting*) *atLeastLessThan-iff map-eq-conv set-upt*)
also have ... = *num* (*take t zs*) + *todigit* (*zs* ! *t*) ∗ *2* ^ *t*
using *num-def True* **by** *simp*
also have ... = *num* (*take t zs*) + *todigit* (*digit zs t*) ∗ *2* ^ *t*
using *digit-def True* **by** *simp*
finally show *?thesis*
by *simp*
next
case *False*
then show *?thesis*
using *digit-def* **by** *simp*
qed

A symbol sequence is a canonical representation of a natural number if the sequence contains only the symbols **0** and **1** and is either empty or ends in **1**.

definition *canonical* :: *symbol list* ⇒ *bool* **where**
canonical xs ≡ *bit-symbols xs* ∧ (*xs* = [] ∨ *last xs* = **1**)

lemma *canonical-Cons*:
assumes *canonical xs* **and** *xs* ≠ [] **and** *x* = **0** ∨ *x* = **1**
shows *canonical* (*x* # *xs*)
using *assms canonical-def less-Suc-eq-0-disj* **by** *auto*

lemma *canonical-Cons-3*: *canonical xs* ⟹ *canonical* (**1** # *xs*)
using *canonical-def less-Suc-eq-0-disj* **by** *auto*

lemma *canonical-tl*: *canonical* (*x* # *xs*) ⟹ *canonical xs*
using *canonical-def* **by** *fastforce*

lemma *prepend-2-even*: *x* = **0** ⟹ *even* (*num* (*x* # *xs*))
using *num-Cons* **by** *simp*

lemma *prepend-3-odd*: *x* = **1** ⟹ *odd* (*num* (*x* # *xs*))
using *num-Cons* **by** *simp*

Every number has exactly one canonical representation.

lemma *canonical-ex1*:
fixes *n* :: *nat*
shows ∃!*xs*. *num xs* = *n* ∧ *canonical xs*
proof (*induction n rule*: *nat-less-induct*)
case *IH*: (*1 n*)
show *?case*
proof (*cases n* = *0*)
case *True*
have *num* [] = *0*
using *num-def* **by** *simp*
moreover have *canonical xs* ⟹ *num xs* = *0* ⟹ *xs* = [] **for** *xs*
proof (*rule ccontr*)
fix *xs*
assume *canonical xs num xs* = *0 xs* ≠ []
then have *length xs* > *0 last xs* = **1**
using *canonical-def* **by** *simp-all*
then have *xs* ! (*length xs* − *1*) = **1**
by (*metis Suc-diff-1 last-length*)

```
    then have num xs ≥ 2 ^ (length xs − 1)
      using num-ge-pow by (meson ‹0 < length xs› diff-less zero-less-one)
    then have num xs > 0
      by (meson dual-order.strict-trans1 le0 le-less-trans less-exp)
    then show False
      using ‹num xs = 0› by auto
  qed
  ultimately show ?thesis
    using IH True canonical-def by (metis less-nat-zero-code list.size(3))
next
  case False
  then have gt: n > 0
    by simp
  define m where m = n div 2
  define r where r = n mod 2
  have n: n = 2 ∗ m + r
    using m-def r-def by simp
  have m < n
    using gt m-def by simp
  then obtain xs where num xs = m canonical xs
    using IH by auto
  then have num (tosym r # xs) = n
      (is num ?xs = n)
    using num-Cons n add.commute r-def by simp
  have canonical ?xs
  proof (cases r = 0)
    case True
    then have m > 0
      using gt n by simp
    then have xs ≠ []
      using ‹num xs = m› num-def by auto
    then show ?thesis
      using canonical-Cons[of xs] ‹canonical xs› r-def True by simp
  next
    case False
    then show ?thesis
      using ‹canonical xs› canonical-Cons-3 r-def
      by (metis One-nat-def not-mod-2-eq-1-eq-0 numeral-3-eq-3 one-add-one plus-1-eq-Suc)
  qed
  moreover have xs1 = xs2 if canonical xs1 num xs1 = n canonical xs2 num xs2 = n for xs1 xs2
  proof −
    have xs1 ≠ []
      using gt that(2) num-def by auto
    then obtain x1 ys1 where 1: xs1 = x1 # ys1
      by (meson neq-Nil-conv)
    then have x1: x1 = 𝟬 ∨ x1 = 𝟭
      using canonical-def that(1) by auto
    have xs2 ≠ []
      using gt that(4) num-def by auto
    then obtain x2 ys2 where 2: xs2 = x2 # ys2
      by (meson neq-Nil-conv)
    then have x2: x2 = 𝟬 ∨ x2 = 𝟭
      using canonical-def that(3) by auto
    have x1 = x2
      using prepend-2-even prepend-3-odd that 1 2 x1 x2 by metis
    moreover have n = todigit x1 + 2 ∗ num ys1
      using that(2) num-Cons 1 by simp
    moreover have n = todigit x2 + 2 ∗ num ys2
      using that(4) num-Cons 2 by simp
    ultimately have num ys1 = num ys2
      by simp
    moreover have num ys1 < n
      using that(2) num-Cons 1 gt by simp
```

moreover have *num ys2* < *n*
using *that*(*4*) *num-Cons 2 gt* **by** *simp*
ultimately have *ys1* = *ys2*
using *IH 1 2 that*(*1*,*3*) **by** (*metis canonical-tl*)
then show *xs1* = *xs2*
using ‹*x1* = *x2*› *1 2* **by** *simp*
qed
ultimately show *?thesis*
using ‹*num* (*tosym r* # *xs*) = *n*› **by** *auto*
qed
qed

The canonical representation of a natural number as symbol sequence:

definition *canrepr* :: *nat* ⇒ *symbol list* **where**
canrepr n ≡ *THE xs. num xs* = *n* ∧ *canonical xs*

lemma *canrepr-inj*: *inj canrepr*
using *canrepr-def canonical-ex1* **by** (*smt* (*verit*, *del-insts*) *inj-def the-equality*)

lemma *canonical-canrepr*: *canonical* (*canrepr n*)
using *theI′*[*OF canonical-ex1*] *canrepr-def* **by** *simp*

lemma *canrepr*: *num* (*canrepr n*) = *n*
using *theI′*[*OF canonical-ex1*] *canrepr-def* **by** *simp*

lemma *bit-symbols-canrepr*: *bit-symbols* (*canrepr n*)
using *canonical-canrepr canonical-def* **by** *simp*

lemma *proper-symbols-canrepr*: *proper-symbols* (*canrepr n*)
using *bit-symbols-canrepr* **by** *fastforce*

lemma *canreprI*: *num xs* = *n* ⟹ *canonical xs* ⟹ *canrepr n* = *xs*
using *canrepr canonical-canrepr canonical-ex1* **by** *blast*

lemma *canrepr-0*: *canrepr 0* = []
using *num-def canonical-def* **by** (*intro canreprI*) *simp-all*

lemma *canrepr-1*: *canrepr 1* = [**1**]
using *num-def canonical-def* **by** (*intro canreprI*) *simp-all*

The length of the canonical representation of a number n:

abbreviation *nlength* :: *nat* ⇒ *nat* **where**
nlength n ≡ *length* (*canrepr n*)

lemma *nlength-0*: *nlength n* = *0* ⟷ *n* = *0*
by (*metis canrepr canrepr-0 length-0-conv*)

corollary *nlength-0-simp* [*simp*]: *nlength 0* = *0*
using *nlength-0* **by** *simp*

lemma *num-replicate2-eq-pow*: *num* (*replicate j* **0** @ [**1**]) = *2* ^ *j*
proof (*induction j*)
case *0*
then show *?case*
using *num-def* **by** *simp*
next
case (*Suc j*)
then show *?case*
using *num-Cons* **by** *simp*
qed

lemma *num-replicate3-eq-pow-minus-1*: *num* (*replicate j* **1**) = *2* ^ *j* − *1*
proof (*induction j*)

```
  case 0
  then show ?case
    using num-def by simp
next
  case (Suc j)
  then have num (replicate (Suc j) 𝟭) = num (𝟭 # replicate j 𝟭)
    by simp
  also have ... = 1 + 2 * (2 ^ j − 1)
    using Suc num-Cons by simp
  also have ... = 1 + 2 * 2 ^ j − 2
   by (metis Nat.add-diff-assoc diff-mult-distrib2 mult-2 mult-le-mono2 nat-1-add-1 one-le-numeral one-le-power)
  also have ... = 2 ^ Suc j − 1
    by simp
  finally show ?case .
qed

lemma nlength-pow2: nlength (2 ^ j) = Suc j
proof −
  define xs :: nat list where xs = replicate j 2 @ [3]
  then have length xs = Suc j
    by simp
  moreover have num xs = 2 ^ j
    using num-replicate2-eq-pow xs-def by simp
  moreover have canonical xs
    using xs-def bit-symbols-append canonical-def by simp
  ultimately show ?thesis
    using canreprI by blast
qed

corollary nlength-1-simp [simp]: nlength 1 = 1
  using nlength-pow2[of 0] by simp

corollary nlength-2: nlength 2 = 2
  using nlength-pow2[of 1] by simp

lemma nlength-pow-minus-1: nlength (2 ^ j − 1) = j
proof −
  define xs :: nat list where xs = replicate j 𝟭
  then have length xs = j
    by simp
  moreover have num xs = 2 ^ j − 1
    using num-replicate3-eq-pow-minus-1 xs-def by simp
  moreover have canonical xs
  proof −
    have bit-symbols xs
      using xs-def by simp
    moreover have last xs = 3 ∨ xs = []
      by (cases j = 0) (simp-all add: xs-def)
    ultimately show ?thesis
      using canonical-def by auto
  qed
  ultimately show ?thesis
    using canreprI by metis
qed

corollary nlength-3: nlength 3 = 2
  using nlength-pow-minus-1[of 2] by simp
```

When handling natural numbers, Turing machines will usually have tape contents of the following form:

```
abbreviation ncontents :: nat ⇒ (nat ⇒ symbol) (‹⌊-⌋N›) where
  ⌊n⌋N ≡ ⌊canrepr n⌋

lemma ncontents-0: ⌊0⌋N = ⌊[]⌋
```

by (*simp add*: *canrepr-0*)

lemma *clean-tape-ncontents*: *clean-tape* ($\lfloor x \rfloor_N$, *i*)
using *bit-symbols-canrepr clean-contents-proper* **by** *fastforce*

lemma *ncontents-1-blank-iff-zero*: $\lfloor n \rfloor_N$ *1* = $\Box \longleftrightarrow$ *n* = *0*
using *bit-symbols-canrepr contents-def nlength-0*
by (*metis contents-outofbounds diff-is-0-eq′ leI length-0-conv length-greater-0-conv less-one zero-neq-numeral*)

Every bit symbol sequence can be turned into a canonical representation of some number by stripping trailing zeros. The length of the prefix without trailing zeros is given by the next function:

definition *canlen* :: *symbol list* $\Rightarrow$ *nat* **where**
canlen zs $\equiv$ *LEAST m*. $\forall$ *i*<*length zs*. *i* $\geq$ *m* $\longrightarrow$ *zs* ! *i* = **0**

lemma *canlen-at-ge*: $\forall$ *i*<*length zs*. *i* $\geq$ *canlen zs* $\longrightarrow$ *zs* ! *i* = **0**
proof −
let *?P* = λ*m*. $\forall$ *i*<*length zs*. *i* $\geq$ *m* $\longrightarrow$ *zs* ! *i* = **0**
have *?P* (*length zs*)
by *simp*
then show *?thesis*
unfolding *canlen-def* **using** *LeastI*[*of ?P length zs*] **by** *fast*
qed

lemma *canlen-eqI*:
assumes $\forall$ *i*<*length zs*. *i* $\geq$ *m* $\longrightarrow$ *zs* ! *i* = **0**
and $\bigwedge$*y*. $\forall$ *i*<*length zs*. *i* $\geq$ *y* $\longrightarrow$ *zs* ! *i* = **0** $\Longrightarrow$ *m* $\leq$ *y*
shows *canlen zs* = *m*
unfolding *canlen-def* **using** *assms Least-equality*[*of - m*, *OF - assms*(*2*)] **by** *presburger*

lemma *canlen-le-length*: *canlen zs* $\leq$ *length zs*
proof −
let *?P* = λ*m*. $\forall$ *i*<*length zs*. *i* $\geq$ *m* $\longrightarrow$ *zs* ! *i* = **0**
have *?P* (*length zs*)
by *simp*
then show *?thesis*
unfolding *canlen-def* **using** *Least-le*[*of - length zs*] **by** *simp*
qed

lemma *canlen-le*:
assumes $\forall$ *i*<*length zs*. *i* $\geq$ *m* $\longrightarrow$ *zs* ! *i* = **0**
shows *m* $\geq$ *canlen zs*
unfolding *canlen-def* **using** *Least-le*[*of - m*] *assms* **by** *simp*

lemma *canlen-one*:
assumes *bit-symbols zs* **and** *canlen zs* > *0*
shows *zs* ! (*canlen zs* − *1*) = **1**
proof (*rule ccontr*)
assume *zs* ! (*canlen zs* − *1*) $\neq$ **1**
then have *zs* ! (*canlen zs* − *1*) = **0**
using *assms canlen-le-length*
by (*metis One-nat-def Suc-pred lessI less-le-trans*)
then have $\forall$ *i*<*length zs*. *i* $\geq$ *canlen zs* − *1* $\longrightarrow$ *zs* ! *i* = *2*
using *canlen-at-ge assms*(*2*) **by** (*metis One-nat-def Suc-leI Suc-pred le-eq-less-or-eq*)
then have *canlen zs* − *1* $\geq$ *canlen zs*
using *canlen-le* **by** *auto*
then show *False*
using *assms*(*2*) **by** *simp*
qed

lemma *canonical-take-canlen*:
assumes *bit-symbols zs*
shows *canonical* (*take* (*canlen zs*) *zs*)
proof (*cases canlen zs* = *0*)

case *True*
then show *?thesis*
using *canonical-def* **by** *simp*
next
case *False*
then show *?thesis*
using *canonical-def assms canlen-le-length canlen-one*
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def Suc-pred append-take-drop-id diff-less last-length length-take less-le-trans min-absorb2 neq0-conv nth-append zero-less-one*)
qed

lemma *num-take-canlen-eq*: *num* (*take* (*canlen zs*) *zs*) = *num zs*
proof (*induction length zs* − *canlen zs arbitrary*: *zs*)
case *0*
then show *?case*
by *simp*
next
case (*Suc x*)
let *?m* = *canlen zs*
have ∗: ∀ *i*<*length zs*. *i* ≥ *?m* ⟶ *zs* ! *i* = **0**
using *canlen-at-ge* **by** *auto*
have *canlen zs* < *length zs*
using *Suc* **by** *simp*
then have *zs* ! (*length zs* − *1*) = **0**
using *Suc canlen-at-ge canlen-le-length*
by (*metis One-nat-def Suc-pred diff-less le-Suc-eq less-nat-zero-code nat-neq-iff zero-less-one*)
then have *todigit* (*zs* ! (*length zs* − *1*)) = *0*
by *simp*
moreover have *ys*: *zs* = *take* (*length zs* − *1*) *zs* @ [*zs* ! (*length zs* − *1*)]
(**is** *zs* = *?ys* @ -)
by (*metis Suc-diff-1* ‹*canlen zs* < *length zs*› *append-butlast-last-id butlast-conv-take gr-implies-not0 last-length length-0-conv length-greater-0-conv*)
ultimately have *num ?ys* = *num zs*
using *num-trailing-zero* **by** *metis*
have *canlen-ys*: *canlen ?ys* = *canlen zs*
proof (*rule canlen-eqI*)
show ∀ *i*<*length ?ys*. *canlen zs* ≤ *i* ⟶ *?ys* ! *i* = **0**
by (*simp add*: *canlen-at-ge*)
show ⋀*y*. ∀ *i*<*length ?ys*. *y* ≤ *i* ⟶ *?ys* ! *i* = **0** ⟹ *canlen zs* ≤ *y*
using ∗ *Suc.hyps*(*2*) *canlen-le*
by (*smt* (*verit*, *del-insts*) *One-nat-def Suc-pred append-take-drop-id diff-le-self length-take length-upt less-Suc-eq less-nat-zero-code list.size*(*3*) *min-absorb2 nth-append upt.simps*(*2*) *zero-less-Suc*)
qed
then have *length ?ys* − *canlen ?ys* = *x*
using *ys Suc.hyps*(*2*) **by** (*metis butlast-snoc diff-Suc-1 diff-commute length-butlast*)
then have *num* (*take* (*canlen ?ys*) *?ys*) = *num ?ys*
using *Suc* **by** *blast*
then have *num* (*take* (*canlen zs*) *?ys*) = *num ?ys*
using *canlen-ys* **by** *simp*
then have *num* (*take* (*canlen zs*) *zs*) = *num ?ys*
by (*metis* ‹*canlen zs* < *length zs*› *butlast-snoc take-butlast ys*)
then show *?case*
using ‹*num ?ys* = *num zs*› **by** *presburger*
qed

lemma *canrepr-take-canlen*:
assumes *num zs* = *n* **and** *bit-symbols zs*
shows *canrepr n* = *take* (*canlen zs*) *zs*
using *assms canrepr canonical-canrepr canonical-ex1 canonical-take-canlen num-take-canlen-eq*
by *blast*

lemma *length-canrepr-canlen*:
assumes *num zs* = *n* **and** *bit-symbols zs*

```
  shows nlength n = canlen zs
  using canrepr-take-canlen assms canlen-le-length by (metis length-take min-absorb2)

lemma nlength-ge-pow:
  assumes nlength n = Suc j
  shows n ≥ 2 ^ j
proof -
  let ?xs = canrepr n
  have ?xs ! (length ?xs - 1) = 𝟭
    using canonical-def assms canonical-canrepr
    by (metis Suc-neq-Zero diff-Suc-1 last-length length-0-conv)
  moreover have (∑i←[0..<length ?xs]. todigit (?xs ! i) * 2 ^ i) ≥
      todigit (?xs ! (length ?xs - 1)) * 2 ^ (length ?xs - 1)
    using assms by simp
  ultimately have num ?xs ≥ 2 ^ (length ?xs - 1)
    using num-def by simp
  moreover have num ?xs = n
    using canrepr by simp
  ultimately show ?thesis
    using assms by simp
qed

lemma nlength-less-pow: n < 2 ^ (nlength n)
proof (induction nlength n arbitrary: n)
  case 0
  then show ?case
    by (metis canrepr canrepr-0 length-0-conv nat-zero-less-power-iff)
next
  case (Suc j)
  let ?xs = canrepr n
  have lenxs: length ?xs = Suc j
    using Suc by simp
  have hdtl: ?xs = hd ?xs # tl ?xs
    using Suc by (metis hd-Cons-tl list.size(3) nat.simps(3))
  have len: length (tl ?xs) = j
    using Suc by simp
  have can: canonical (tl ?xs)
    using hdtl canonical-canrepr canonical-tl by metis
  define n' where n' = num (tl ?xs)
  then have nlength n' = j
    using len can canreprI by simp
  then have n'-less: n' < 2 ^ j
    using Suc by auto
  have num ?xs = todigit (hd ?xs) + 2 * num (tl ?xs)
    by (metis hdtl num-Cons)
  then have n = todigit (hd ?xs) + 2 * num (tl ?xs)
    using canrepr by simp
  also have ... ≤ 1 + 2 * num (tl ?xs)
    by simp
  also have ... = 1 + 2 * n'
    using n'-def by simp
  also have ... ≤ 1 + 2 * (2 ^ j - 1)
    using n'-less by simp
  also have ... = 2 ^ (Suc j) - 1
    by (metis (no-types, lifting) add-Suc-right le-add-diff-inverse mult-2 one-le-numeral
      one-le-power plus-1-eq-Suc sum.op-ivl-Suc sum-power2 zero-order(3))
  also have ... < 2 ^ (Suc j)
    by simp
  also have ... = 2 ^ (nlength n)
    using lenxs by simp
  finally show ?case .
qed
```

```
lemma pow-nlength:
  assumes 2 ^ j ≤ n and n < 2 ^ (Suc j)
  shows nlength n = Suc j
proof (rule ccontr)
  assume nlength n ≠ Suc j
  then have nlength n < Suc j ∨ nlength n > Suc j
    by auto
  then show False
  proof
    assume nlength n < Suc j
    then have nlength n ≤ j
      by simp
    moreover have n < 2 ^ (nlength n)
      using nlength-less-pow by simp
    ultimately have n < 2 ^ j
      by (metis le-less-trans nat-power-less-imp-less not-less numeral-2-eq-2 zero-less-Suc)
    then show False
      using assms(1) by simp
  next
    assume *: nlength n > Suc j
    then have n ≥ 2 ^ (nlength n − 1)
      using nlength-ge-pow by simp
    moreover have nlength n − 1 ≥ Suc j
      using * by simp
    ultimately have n ≥ 2 ^ (Suc j)
      by (metis One-nat-def le-less-trans less-2-cases-iff linorder-not-less power-less-imp-less-exp)
    then show False
      using assms(2) by simp
  qed
qed

lemma nlength-le-n: nlength n ≤ n
proof (cases n = 0)
  case True
  then show ?thesis
    using canrepr-0 by simp
next
  case False
  then have nlength n > 0
    using nlength-0 by simp
  moreover from this have n ≥ 2 ^ (nlength n − 1)
    using nlength-0 nlength-ge-pow by auto
  ultimately show ?thesis
    using nlength-ge-pow by (metis Suc-diff-1 Suc-leI dual-order.trans less-exp)
qed

lemma nlength-Suc-le: nlength n ≤ nlength (Suc n)
proof (cases n = 0)
  case True
  then show ?thesis
    by (simp add: canrepr-0)
next
  case False
  then obtain j where j: nlength n = Suc j
    by (metis canrepr canrepr-0 gr0-implies-Suc length-greater-0-conv)
  then have n ≥ 2 ^ j
    using nlength-ge-pow by simp
  show ?thesis
  proof (cases Suc n ≥ 2 ^ (Suc j))
    case True
    have n < 2 ^ (Suc j)
      using j nlength-less-pow by metis
    then have Suc n < 2 ^ (Suc (Suc j))
```

```
      by simp
    then have nlength (Suc n) = Suc (Suc j)
      using True pow-nlength by simp
    then show ?thesis
      using j by simp
  next
    case False
    then have Suc n < 2 ^ (Suc j)
      by simp
    then have nlength (Suc n) = Suc j
      using ‹n ≥ 2 ^ j› pow-nlength by simp
    then show ?thesis
      using j by simp
  qed
qed

lemma nlength-mono:
  assumes n1 ≤ n2
  shows nlength n1 ≤ nlength n2
proof −
  have nlength n ≤ nlength (n + d) for n d
  proof (induction d)
    case 0
    then show ?case
      by simp
  next
    case (Suc d)
    then show ?case
      using nlength-Suc-le by (metis nat-arith.suc1 order-trans)
  qed
  then show ?thesis
    using assms by (metis le-add-diff-inverse)
qed

lemma nlength-even-le: n > 0 ⟹ nlength (2 ∗ n) = Suc (nlength n)
proof −
  assume n > 0
  then have nlength n > 0
    by (metis canrepr canrepr-0 length-greater-0-conv less-numeral-extra(3))
  then have n ≥ 2 ^ (nlength n − 1)
    using Suc-diff-1 nlength-ge-pow by simp
  then have 2 ∗ n ≥ 2 ^ (nlength n)
    by (metis Suc-diff-1 ‹0 < nlength n› mult-le-mono2 power-Suc)
  moreover have 2 ∗ n < 2 ^ (Suc (nlength n))
    using nlength-less-pow by simp
  ultimately show ?thesis
    using pow-nlength by simp
qed

lemma nlength-prod: nlength (n1 ∗ n2) ≤ nlength n1 + nlength n2
proof −
  let ?j1 = nlength n1 and ?j2 = nlength n2
  have n1 < 2 ^ ?j1 n2 < 2 ^ ?j2
    using nlength-less-pow by simp-all
  then have n1 ∗ n2 < 2 ^ ?j1 ∗ 2 ^ ?j2
    by (simp add: mult-strict-mono)
  then have n1 ∗ n2 < 2 ^ (?j1 + ?j2)
    by (simp add: power-add)
  then have n1 ∗ n2 ≤ 2 ^ (?j1 + ?j2) − 1
    by simp
  then have nlength (n1 ∗ n2) ≤ nlength (2 ^ (?j1 + ?j2) − 1)
    using nlength-mono by simp
  then show nlength (n1 ∗ n2) ≤ ?j1 + ?j2
```

    using *nlength-pow-minus-1* **by** *simp*
qed

In the following lemma *Suc* is needed because $n^0 = 1$.

lemma *nlength-pow*: $nlength\ (n\ \hat{}\ d) \le Suc\ (d * nlength\ n)$
proof (*induction d*)
  case *0*
  then show *?case*
    by (*metis less-or-eq-imp-le mult-not-zero nat-power-eq-Suc-0-iff nlength-pow2*)
next
  case (*Suc d*)
  have $nlength\ (n\ \hat{}\ Suc\ d) = nlength\ (n\ \hat{}\ d * n)$
    by (*simp add*: *mult.commute*)
  then have $nlength\ (n\ \hat{}\ Suc\ d) \le nlength\ (n\ \hat{}\ d) + nlength\ n$
    using *nlength-prod* **by** *simp*
  then show *?case*
    using *Suc* **by** *simp*
qed

lemma *nlength-sum*: $nlength\ (n1 + n2) \le Suc\ (max\ (nlength\ n1)\ (nlength\ n2))$
proof −
  let $?m = max\ n1\ n2$
  have $n1 + n2 \le 2 * ?m$
    by *simp*
  then have $nlength\ (n1 + n2) \le nlength\ (2 * ?m)$
    using *nlength-mono* **by** *simp*
  moreover have $nlength\ ?m = max\ (nlength\ n1)\ (nlength\ n2)$
    using *nlength-mono* **by** (*metis max.absorb1 max.cobounded2 max-def*)
  ultimately show *?thesis*
    using *nlength-even-le*
    by (*metis canrepr-0 le-SucI le-zero-eq list.size*(*3*) *max-nat.neutr-eq-iff not-gr-zero zero-eq-add-iff-both-eq-0*)
qed

lemma *nlength-Suc*: $nlength\ (Suc\ n) \le Suc\ (nlength\ n)$
  using *nlength-sum nlength-1-simp*
  by (*metis One-nat-def Suc-leI add-Suc diff-Suc-1 length-greater-0-conv max.absorb-iff2 max.commute max-def nlength-0 plus-1-eq-Suc*)

lemma *nlength-less-n*: $n \ge 3 \Longrightarrow nlength\ n < n$
proof (*induction n rule*: *nat-induct-at-least*)
  case *base*
  then show *?case*
    by (*simp add*: *nlength-3*)
next
  case (*Suc n*)
  then show *?case*
    using *nlength-Suc* **by** (*metis Suc-le-eq le-neq-implies-less nlength-le-n not-less-eq*)
qed

Comparing two numbers

In order to compare two numbers in canonical representation, we can use the Turing machine *tm-equals*, which works for arbitrary proper symbol sequences.

lemma *min-nlength*: $min\ (nlength\ n1)\ (nlength\ n2) = nlength\ (min\ n1\ n2)$
  by (*metis min-absorb2 min-def nat-le-linear nlength-mono*)

lemma *max-nlength*: $max\ (nlength\ n1)\ (nlength\ n2) = nlength\ (max\ n1\ n2)$
  using *nlength-mono* **by** (*metis max.absorb1 max.cobounded2 max-def*)

lemma *contents-blank-0*: $\lfloor[\Box]\rfloor = \lfloor[]\rfloor$
  using *contents-def* **by** *auto*

definition *tm-equalsn* :: *tapeidx* ⇒ *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-equalsn ≡ *tm-equals*

lemma *tm-equalsn-tm*:
assumes *k* ≥ *2* **and** *G* ≥ *4* **and** *0* < *j3*
shows *turing-machine k G* (*tm-equalsn j1 j2 j3*)
unfolding *tm-equalsn-def* **using** *assms tm-equals-tm* **by** *simp*

lemma *transforms-tm-equalsnI* [*transforms-intros*]:
fixes *j1 j2 j3* :: *tapeidx*
fixes *tps tps'* :: *tape list* **and** *k b n1 n2* :: *nat*
assumes *length tps* = *k j1* ≠ *j2 j2* ≠ *j3 j1* ≠ *j3 j1* < *k j2* < *k j3* < *k*
and *b* ≤ *1*
assumes
tps ! *j1* = (⌊*n1*⌋$_N$, *1*)
tps ! *j2* = (⌊*n2*⌋$_N$, *1*)
tps ! *j3* = (⌊*b*⌋$_N$, *1*)
assumes *ttt* = (*3* ∗ *nlength* (*min n1 n2*) + *7*)
assumes *tps'* = *tps*
[*j3* := (⌊*if n1* = *n2 then 1 else 0*⌋$_N$, *1*)]
shows *transforms* (*tm-equalsn j1 j2 j3*) *tps ttt tps'*
unfolding *tm-equalsn-def*
proof (*tform tps*: *assms*)
show *proper-symbols* (*canrepr n1*)
using *proper-symbols-canrepr* **by** *simp*
show *proper-symbols* (*canrepr n2*)
using *proper-symbols-canrepr* **by** *simp*
show *ttt* = *3* ∗ *min* (*nlength n1*) (*nlength n2*) + *7*
using *assms*(*12*) *min-nlength* **by** *simp*
let *?v* = *if canrepr n1* = *canrepr n2 then 3*::*nat else 0*::*nat*
have *b* = *0* ∨ *b* = *1*
using *assms*(*8*) **by** *auto*
then have ⌊*b*⌋$_N$ = ⌊[]⌋ ∨ ⌊*b*⌋$_N$ = ⌊[**1**]⌋
using *canrepr-0 canrepr-1* **by** *auto*
then have *tps* ! *j3* = (⌊[]⌋, *1*) ∨ *tps* ! *j3* = (⌊[**1**]⌋, *1*)
using *assms*(*11*) **by** *simp*
then have *v*: *tps* ! *j3* |:=| *?v* = (⌊[*?v*]⌋, *1*)
using *contents-def* **by** *auto*
show *tps'* = *tps*[*j3* := *tps* ! *j3* |:=| *?v*]
proof (*cases n1* = *n2*)
case *True*
then show *?thesis*
using *canrepr-1 v assms*(*13*) **by** *auto*
next
case *False*
then have *?v* = *0*
by (*metis canrepr*)
then show *?thesis*
using *canrepr-0 v assms*(*13*) *contents-blank-0* **by** *auto*
qed
qed

Copying a number between tapes

The next Turing machine overwrites the contents of tape j_2 with the contents of tape j_1 and performs a carriage return on both tapes.

definition *tm-copyn* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-copyn j1 j2 ≡
tm-erase-cr j2 ;;
tm-cp-until j1 j2 {□} ;;
tm-cr j1 ;;
tm-cr j2

lemma *tm-copyn-tm*:
  assumes $k \geq 2$ **and** $G \geq 4$ **and** $j1 < k$ $j2 < k$ $j1 \neq j2$ $0 < j2$
  shows *turing-machine k G* (*tm-copyn j1 j2*)
  unfolding *tm-copyn-def* **using** *assms tm-cp-until-tm tm-cr-tm tm-erase-cr-tm* **by** *simp*

locale *turing-machine-move* =
  fixes *j1 j2* :: *tapeidx*
begin

definition *tm1* ≡ *tm-erase-cr j2*
definition *tm2* ≡ *tm1* ;; *tm-cp-until j1 j2* {□}
definition *tm3* ≡ *tm2* ;; *tm-cr j1*
definition *tm4* ≡ *tm3* ;; *tm-cr j2*

lemma *tm4-eq-tm-copyn*: *tm4* = *tm-copyn j1 j2*
  unfolding *tm4-def tm3-def tm2-def tm1-def tm-copyn-def* **by** *simp*

context
  fixes *x y* :: *nat* **and** *tps0* :: *tape list*
  assumes *j-less* [*simp*]: $j1 < length\ tps0$ $j2 < length\ tps0$
  assumes *j* [*simp*]: $j1 \neq j2$
    and *tps-j1* [*simp*]: *tps0* ! *j1* = ($\lfloor x \rfloor_N$, *1*)
    and *tps-j2* [*simp*]: *tps0* ! *j2* = ($\lfloor y \rfloor_N$, *1*)
begin

definition *tps1* ≡ *tps0*
  [*j2* := (⌊[]⌋, *1*)]

lemma *tm1* [*transforms-intros*]:
  assumes $t = 7 + 2 * nlength\ y$
  shows *transforms tm1 tps0 t tps1*
  unfolding *tm1-def*
proof (*tform tps*: *tps1-def time*: *assms*)
  show *proper-symbols* (*canrepr y*)
    using *proper-symbols-canrepr* **by** *simp*
qed

definition *tps2* ≡ *tps0*
  [*j1* := ($\lfloor x \rfloor_N$, *Suc* (*nlength x*)),
   j2 := ($\lfloor x \rfloor_N$, *Suc* (*nlength x*))]

lemma *tm2* [*transforms-intros*]:
  assumes $t = 8 + (2 * nlength\ y + nlength\ x)$
  shows *transforms tm2 tps0 t tps2*
  unfolding *tm2-def*
proof (*tform tps*: *tps1-def time*: *assms*)
  show *rneigh* (*tps1* ! *j1*) {□} (*nlength x*)
  proof (*rule rneighI*)
    show (*tps1* ::: *j1*) (*tps1* :#: *j1* + *nlength x*) ∈ {□}
      using *tps1-def canrepr-0 contents-outofbounds j*(*1*) *nlength-0-simp tps-j1*
      by (*metis fst-eqD lessI nth-list-update-neq plus-1-eq-Suc singleton-iff snd-eqD*)
    show $\bigwedge n'.\ n' < nlength\ x \Longrightarrow$ (*tps1* ::: *j1*) (*tps1* :#: *j1* + n') ∉ {□}
      using *tps1-def tps-j1 j j-less contents-inbounds proper-symbols-canrepr*
      by (*metis Suc-leI add-diff-cancel-left′ fst-eqD not-add-less2 nth-list-update-neq*
        plus-1-eq-Suc singletonD snd-eqD zero-less-Suc)
  qed

  have ($\lfloor x \rfloor_N$, *Suc* (*nlength x*)) = *tps0*[*j2* := (⌊[]⌋, *1*)] ! *j1* |+| *nlength x*
    using *tps-j1 tps-j2* **by** (*metis fst-eqD j*(*1*) *j-less*(*2*) *nth-list-update plus-1-eq-Suc snd-eqD*)
  moreover have ($\lfloor x \rfloor_N$, *Suc* (*nlength x*)) =
      implant (*tps0*[*j2* := (⌊[]⌋, *1*)] ! *j1*) (*tps0*[*j2* := (⌊[]⌋, *1*)] ! *j2*) (*nlength x*)
    using *tps-j1 tps-j2 j j-less implant-contents nlength-0-simp*
    by (*metis add.right-neutral append.simps*(*1*) *canrepr-0 diff-Suc-1 drop0 le-eq-less-or-eq*

nth-list-update-eq nth-list-update-neq plus-1-eq-Suc take-all zero-less-one)
ultimately show *tps2* = *tps1*
[*j1* := *tps1* ! *j1* |+| *nlength x*,
j2 := *implant* (*tps1* ! *j1*) (*tps1* ! *j2*) (*nlength x*)]
unfolding *tps2-def tps1-def* **by** (*simp add*: *list-update-swap*[*of j1*])
qed

definition *tps3* ≡ *tps0*[*j2* := (⌊*x*⌋$_N$, *Suc* (*nlength x*))]

lemma *tm3* [*transforms-intros*]:
assumes *t* = *11* + (*2* ∗ *nlength y* + *2* ∗ *nlength x*)
shows *transforms tm3 tps0 t tps3*
unfolding *tm3-def*
proof (*tform tps*: *tps2-def*)
have *tps2* :#: *j1* = *Suc* (*nlength x*)
using *assms tps2-def* **by** (*metis j*(*1*) *j-less*(*1*) *nth-list-update-eq nth-list-update-neq snd-conv*)
then show *t* = *8* + (*2* ∗ *nlength y* + *nlength x*) + (*tps2* :#: *j1* + *2*)
using *assms* **by** *simp*
show *clean-tape* (*tps2* ! *j1*)
using *tps2-def* **by** (*simp add*: *clean-tape-ncontents nth-list-update-neq′*)
have *tps2* ! *j1* |#=| *1* = (⌊*x*⌋$_N$, *1*)
using *tps2-def* **by** (*simp add*: *nth-list-update-neq′*)
then show *tps3* = *tps2*[*j1* := *tps2* ! *j1* |#=| *1*]
using *tps3-def tps2-def* **by** (*metis j*(*1*) *list-update-id list-update-overwrite list-update-swap tps-j1*)
qed

definition *tps4* ≡ *tps0*[*j2* := (⌊*x*⌋$_N$, *1*)]

lemma *tm4*:
assumes *t* = *14* + (*3* ∗ *nlength x* + *2* ∗ *nlength y*)
shows *transforms tm4 tps0 t tps4*
unfolding *tm4-def*
proof (*tform tps*: *tps3-def time*: *tps3-def assms*)
show *clean-tape* (*tps3* ! *j2*)
using *tps3-def clean-tape-ncontents* **by** *simp*
have *tps3* ! *j2* |#=| *1* = (⌊*x*⌋$_N$, *1*)
using *tps3-def* **by** (*simp add*: *nth-list-update-neq′*)
then show *tps4* = *tps3*[*j2* := *tps3* ! *j2* |#=| *1*]
using *tps4-def tps3-def* **by** (*metis list-update-overwrite tps-j1*)
qed

lemma *tm4′*:
assumes *t* = *14* + *3* ∗ (*nlength x* + *nlength y*)
shows *transforms tm4 tps0 t tps4*
using *tm4 transforms-monotone assms* **by** *simp*

end

end

lemma *transforms-tm-copynI* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx*
fixes *tps tps′* :: *tape list* **and** *k x y* :: *nat*
assumes *j1* ≠ *j2 j1* < *length tps j2* < *length tps*
assumes
tps ! *j1* = (⌊*x*⌋$_N$, *1*)
tps ! *j2* = (⌊*y*⌋$_N$, *1*)
assumes *ttt* = *14* + *3* ∗ (*nlength x* + *nlength y*)
assumes *tps′* = *tps*
[*j2* := (⌊*x*⌋$_N$, *1*)]
shows *transforms* (*tm-copyn j1 j2*) *tps ttt tps′*
proof −
interpret *loc*: *turing-machine-move j1 j2* **.**

show *?thesis*
  using *assms loc.tm4′ loc.tps4-def loc.tm4-eq-tm-copyn* **by** *simp*
qed

Setting the tape contents to a number

The Turing machine in this section writes a hard-coded number to a tape.

definition *tm-setn* :: *tapeidx* ⇒ *nat* ⇒ *machine* **where**
 tm-setn j n ≡ *tm-set j* (*canrepr n*)

lemma *tm-setn-tm*:
 assumes $k \geq 2$ **and** $G \geq 4$ **and** $j < k$ **and** $0 < j$
 shows *turing-machine k G* (*tm-setn j n*)
proof −
 have *symbols-lt G* (*canrepr n*)
  using *assms*(*2*) *bit-symbols-canrepr* **by** *fastforce*
 then show *?thesis*
  unfolding *tm-setn-def* **using** *tm-set-tm assms* **by** *simp*
qed

lemma *transforms-tm-setnI* [*transforms-intros*]:
 fixes *j* :: *tapeidx*
 fixes *tps tps′* :: *tape list* **and** *x k n* :: *nat*
 assumes *j* < *length tps*
 assumes *tps* ! *j* = $(\lfloor x \rfloor_N, 1)$
 assumes $t = 10 + 2 * nlength\ x + 2 * nlength\ n$
 assumes *tps′* = *tps*[*j* := $(\lfloor n \rfloor_N, 1)$]
 shows *transforms* (*tm-setn j n*) *tps t tps′*
 unfolding *tm-setn-def*
 using *transforms-tm-setI*[*OF assms*(*1*), *of canrepr x canrepr n t tps′*] *assms*
  canonical-canrepr canonical-def contents-clean-tape′
 by (*simp add*: *eval-nat-numeral*(*3*) *numeral-Bit0 proper-symbols-canrepr*)

2.7.2 Incrementing

In this section we devise a Turing machine that increments a number. The next function describes how the symbol sequence of the incremented number looks like. Basically one has to flip all **1** symbols starting at the least significant digit until one reaches a **0**, which is then replaced by a **1**. If there is no **0**, a **1** is appended. Here we exploit that the most significant digit is to the right.

definition *nincr* :: *symbol list* ⇒ *symbol list* **where**
 nincr zs ≡
  if ∃*i*<*length zs. zs* ! *i* = **0**
  then replicate (*LEAST i. i* < *length zs* ∧ *zs* ! *i* = **0**) **0** @ [**1**] @ *drop* (*Suc* (*LEAST i. i* < *length zs* ∧ *zs* ! *i* = **0**)) *zs*
  else replicate (*length zs*) **0** @ [**1**]

lemma *canonical-nincr*:
 assumes *canonical zs*
 shows *canonical* (*nincr zs*)
proof −
 have *1*: *bit-symbols zs*
  using *canonical-def assms* **by** *simp*
 let *?j* = *LEAST i. i* < *length zs* ∧ *zs* ! *i* = **0**
 have *bit-symbols* (*nincr zs*)
 proof (*cases* ∃*i*<*length zs. zs* ! *i* = **0**)
  case *True*
  then have *nincr zs* = *replicate ?j* **0** @ [**1**] @ *drop* (*Suc ?j*) *zs*
   using *nincr-def* **by** *simp*
  moreover have *bit-symbols* (*replicate ?j* **0**)
   by *simp*
  moreover have *bit-symbols* [**1**]
   by *simp*

```
    moreover have bit-symbols (drop (Suc ?j) zs)
      using 1 by simp
    ultimately show ?thesis
      using bit-symbols-append by presburger
  next
    case False
    then show ?thesis
      using nincr-def bit-symbols-append by auto
  qed
  moreover have last (nincr zs) = 𝟭
  proof (cases ∃ i<length zs. zs ! i = 𝟬)
    case True
    then show ?thesis
      using nincr-def assms canonical-def by auto
  next
    case False
    then show ?thesis
      using nincr-def by auto
  qed
  ultimately show ?thesis
    using canonical-def by simp
qed

lemma nincr:
  assumes bit-symbols zs
  shows num (nincr zs) = Suc (num zs)
proof (cases ∃ i<length zs. zs ! i = 𝟬)
  case True
  define j where j = (LEAST i. i < length zs ∧ zs ! i = 𝟬)
  then have 1: j < length zs ∧ zs ! j = 𝟬
    using LeastI-ex[OF True] by simp
  have 2: zs ! i = 𝟭 if i < j for i
    using that True j-def assms 1 less-trans not-less-Least by blast

  define xs :: symbol list where xs = replicate j 𝟭 @ [𝟬]
  define ys :: symbol list where ys = drop (Suc j) zs
  have zs = xs @ ys
  proof -
    have xs = take (Suc j) zs
      using xs-def 1 2
      by (smt (verit, best) le-eq-less-or-eq length-replicate length-take min-absorb2 nth-equalityI
       nth-replicate nth-take take-Suc-conv-app-nth)
    then show ?thesis
      using ys-def by simp
  qed

  have nincr zs = replicate j 𝟬 @ [𝟭] @ drop (Suc j) zs
    using nincr-def True j-def by simp
  then have num (nincr zs) = num (replicate j 𝟬 @ [𝟭] @ ys)
    using ys-def by simp
  also have ... = num (replicate j 𝟬 @ [𝟭]) + 2 ^ Suc j * num ys
    using num-append by (metis append-assoc length-append-singleton length-replicate)
  also have ... = Suc (num xs) + 2 ^ Suc j * num ys
  proof -
    have num (replicate j 𝟬 @ [𝟭]) = 2 ^ j
      using num-replicate2-eq-pow by simp
    also have ... = Suc (2 ^ j - 1)
      by simp
    also have ... = Suc (num (replicate j 𝟭))
      using num-replicate3-eq-pow-minus-1 by simp
    also have ... = Suc (num (replicate j 𝟭 @ [𝟬]))
      using num-trailing-zero by simp
    finally have num (replicate j 𝟬 @ [𝟭]) = Suc (num xs)
```

using *xs-def* **by** *simp*
then show *?thesis*
by *simp*
qed
also have ... $= Suc\ (num\ xs + 2\ \hat{}\ Suc\ j * num\ ys)$
by *simp*
also have ... $= Suc\ (num\ zs)$
using ‹$zs = xs\ @\ ys$› *num-append xs-def* **by** (*metis length-append-singleton length-replicate*)
finally show *?thesis* **.**
next
case *False*
then have $\forall i<length\ zs.\ zs\ !\ i = \mathbf{1}$
using *assms* **by** *simp*
then have *zs*: $zs = replicate\ (length\ zs)\ \mathbf{1}$
by (*simp add*: *nth-equalityI*)
then have *num-zs*: $num\ zs = 2\ \hat{}\ length\ zs - 1$
by (*metis num-replicate3-eq-pow-minus-1*)
have $nincr\ zs = replicate\ (length\ zs)\ \mathbf{0}\ @\ [\mathbf{1}]$
using *nincr-def False* **by** *auto*
then have $num\ (nincr\ zs) = 2\ \hat{}\ length\ zs$
by (*simp add*: *num-replicate2-eq-pow*)
then show *?thesis*
using *num-zs* **by** *simp*
qed

lemma *nincr-canrepr*: $nincr\ (canrepr\ n) = canrepr\ (Suc\ n)$
using *canrepr canonical-canrepr canreprI bit-symbols-canrepr canonical-nincr nincr*
by *metis*

The next Turing machine performs the incrementing. Starting from the left of the symbol sequence on tape j, it writes the symbol **0** until it reaches a blank or the symbol **1**. Then it writes a **1** and returns the tape head to the beginning.

definition *tm-incr* :: *tapeidx* ⇒ *machine* **where**
$tm\text{-}incr\ j \equiv tm\text{-}const\text{-}until\ j\ j\ \{\Box, \mathbf{0}\}\ \mathbf{0}\ ;;\ tm\text{-}write\ j\ \mathbf{1}\ ;;\ tm\text{-}cr\ j$

lemma *tm-incr-tm*:
assumes $G \geq 4$ **and** $k \geq 2$ **and** $j < k$ **and** $j > 0$
shows *turing-machine k G* (*tm-incr j*)
unfolding *tm-incr-def* **using** *assms tm-const-until-tm tm-write-tm tm-cr-tm* **by** *simp*

locale *turing-machine-incr* =
fixes *j* :: *tapeidx*
begin

definition $tm1 \equiv tm\text{-}const\text{-}until\ j\ j\ \{\Box, \mathbf{0}\}\ \mathbf{0}$
definition $tm2 \equiv tm1\ ;;\ tm\text{-}write\ j\ \mathbf{1}$
definition $tm3 \equiv tm2\ ;;\ tm\text{-}cr\ j$

lemma *tm3-eq-tm-incr*: $tm3 = tm\text{-}incr\ j$
unfolding *tm3-def tm2-def tm1-def tm-incr-def* **by** *simp*

context
fixes *x k* :: *nat* **and** *tps* :: *tape list*
assumes *jk* [*simp*]: $j < k$ $length\ tps = k$
and *tps0* [*simp*]: $tps\ !\ j = (\lfloor x \rfloor_N,\ 1)$
begin

lemma *tm1* [*transforms-intros*]:
assumes $i0 = (LEAST\ i.\ i \leq nlength\ x \wedge \lfloor x \rfloor_N\ (Suc\ i) \in \{\Box, \mathbf{0}\})$
and $tps' = tps[j := constplant\ (tps\ !\ j)\ \mathbf{0}\ i0]$
shows *transforms tm1 tps* (*Suc i0*) *tps'*
unfolding *tm1-def*
proof (*tform tps*: *assms*(*2*))

```
  let ?P = λi. i ≤ nlength x ∧ ⌊x⌋N (Suc i) ∈ {□, 0}
  have 2: i0 ≤ nlength x ∧ ⌊x⌋N (Suc i0) ∈ {□, 0}
    using LeastI[of ?P nlength x] jk(1) assms(1) by simp
  have 3: ¬ ?P i if i < i0 for i
    using not-less-Least[of i ?P] jk(1) assms(1) that by simp
  show rneigh (tps ! j) {□, 0} i0
  proof (rule rneighI)
    show (tps ::: j) (tps :#: j + i0) ∈ {□, 0}
      using tps0 2 jk(1) assms(1) by simp
    show ⋀n'. n' < i0 ⟹ (tps ::: j) (tps :#: j + n') ∉ {□, 0}
      using tps0 2 3 jk(1) assms(1) by simp
  qed
qed

lemma tm2 [transforms-intros]:
  assumes i0 = (LEAST i. i ≤ nlength x ∧ ⌊x⌋N (Suc i) ∈ {□, 0})
    and ttt = Suc (Suc i0)
    and tps' = tps[j := (⌊Suc x⌋N, Suc i0)]
  shows transforms tm2 tps ttt tps'
  unfolding tm2-def
proof (tform tps: assms(1,3) time: assms(1,2))
  let ?P = λi. i ≤ nlength x ∧ ⌊x⌋N (Suc i) ∈ {□, 0}
  have 1: ?P (nlength x)
    by simp
  have 2: i0 ≤ nlength x ∧ ⌊x⌋N (Suc i0) ∈ {□, 0}
    using LeastI[of ?P nlength x] assms(1) by simp
  have 3: ¬ ?P i if i < i0 for i
    using not-less-Least[of i ?P] assms(1) that by simp
  let ?i = LEAST i. i ≤ nlength x ∧ ⌊x⌋N (Suc i) ∈ {□, 0}
  show tps' = tps
    [j := constplant (tps ! j) 2 ?i,
     j := tps[j := constplant (tps ! j) 0 ?i] ! j |:=| 1]
     (is tps' = ?rhs)
  proof -
    have ?rhs = tps [j := constplant (⌊x⌋N, Suc 0) 0 i0 |:=| 1]
      using jk assms(1) by simp
    moreover have (⌊Suc x⌋N, Suc i0) = constplant (⌊x⌋N, Suc 0) 2 i0 |:=| 1
      (is ?l = ?r)
    proof -
      have snd ?l = snd ?r
        by (simp add: transplant-def)
      moreover have ⌊Suc x⌋N = fst ?r
      proof -
        let ?zs = canrepr x
        have l: ⌊Suc x⌋N = ⌊nincr ?zs⌋
          by (simp add: nincr-canrepr)
        have r: fst ?r = (λi. if Suc 0 ≤ i ∧ i < Suc i0 then 0 else ⌊x⌋N i)(Suc i0 := 1)
          using constplant by auto
        show ?thesis
        proof (cases ∃i<length ?zs. ?zs ! i = 0)
          case True
          let ?Q = λi. i < length ?zs ∧ ?zs ! i = 0
          have Q1: ?Q (Least ?Q)
            using True by (metis (mono-tags, lifting) LeastI-ex)
          have Q2: ¬ ?Q i if i < Least ?Q for i
            using True not-less-Least that by blast
          have Least ?P = Least ?Q
          proof (rule Least-equality)
            show Least ?Q ≤ nlength x ∧ ⌊x⌋N (Suc (Least ?Q)) ∈ {□, 0}
            proof
              show Least ?Q ≤ nlength x
                using True by (metis (mono-tags, lifting) LeastI-ex less-imp-le)
              show ⌊x⌋N (Suc (Least ?Q)) ∈ {□, 0}
```

```
          using True by (simp add: Q1 Suc-leI)
        qed
        then show ⋀y. y ≤ nlength x ∧ ⌊x⌋N (Suc y) ∈ {□, 0} ⟹ (Least ?Q) ≤ y
          using True Q1 Q2 bit-symbols-canrepr contents-def
          by (metis (no-types, lifting) Least-le antisym-conv2 diff-Suc-1 insert-iff
              le-less-Suc-eq less-nat-zero-code nat-le-linear proper-symbols-canrepr singletonD)
      qed
      then have i0: i0 = Least ?Q
        using assms(1) by simp
      then have nincr-zs: nincr ?zs = replicate i0 0 @ [1] @ drop (Suc i0) ?zs
        using nincr-def True by simp
      show ?thesis
      proof
        fix i
        consider
            i = 0
          | Suc 0 ≤ i ∧ i < Suc i0
          | i = Suc i0
          | i > Suc i0 ∧ i ≤ length ?zs
          | i > Suc i0 ∧ i > length ?zs
          by linarith
        then have ⌊replicate i0 0 @ [1] @ drop (Suc i0) ?zs⌋ i =
            ((λi. if Suc 0 ≤ i ∧ i < Suc i0 then 0 else ⌊x⌋N i)(Suc i0 := 1)) i
            (is ?A i = ?B i)
        proof (cases)
          case 1
          then show ?thesis
            by (simp add: transplant-def)
        next
          case 2
          then have i − 1 < i0
            by auto
          then have (replicate i0 0 @ [1] @ drop (Suc i0) ?zs) ! (i − 1) = 0
            by (metis length-replicate nth-append nth-replicate)
          then have ?A i = 0
            using contents-def i0 2 Q1 nincr-canrepr nincr-zs
            by (metis Suc-le-lessD le-trans less-Suc-eq-le less-imp-le-nat less-numeral-extra(3) nlength-Suc-le)
          moreover have ?B i = 0
            using i0 2 by simp
          ultimately show ?thesis
            by simp
        next
          case 3
          then show ?thesis
               using i0 Q1 canrepr-0 contents-inbounds nincr-canrepr nincr-zs nlength-0-simp nlength-Suc
nlength-Suc-le
            by (metis (no-types, lifting) contents-append-update fun-upd-apply length-replicate)
        next
          case 4
          then have ?A i = (replicate i0 0 @ [1] @ drop (Suc i0) ?zs) ! (i − 1)
            by auto
          then have ?A i = ((replicate i0 0 @ [1]) @ drop (Suc i0) ?zs) ! (i − 1)
            by simp
          moreover have length (replicate i0 0 @ [1]) = Suc i0
            by simp
          moreover have i − 1 < length ?zs
            using 4 by auto
          moreover have i − 1 >= Suc i0
            using 4 by auto
          ultimately have ?A i = ?zs ! (i − 1)
            using i0 Q1
           by (metis (no-types, lifting) Suc-leI append-take-drop-id length-take min-absorb2 not-le nth-append)
          moreover have ?B i = ⌊x⌋N i
```

```
        using 4 by simp
      ultimately show ?thesis
        using i0 4 contents-def by simp
    next
      case 5
      then show ?thesis
        by auto
    qed
    then show ⌊Suc x⌋N i = fst (constplant (⌊x⌋N, Suc 0) 0 i0 |:=| 1) i
      using nincr-zs l r by simp
  qed
next
  case False
  then have nincr-zs: nincr ?zs = replicate (length ?zs) 0 @ [1]
    using nincr-def by auto
  have Least ?P = length ?zs
  proof (rule Least-equality)
    show nlength x ≤ nlength x ∧ ⌊x⌋N (Suc (nlength x)) ∈ {□, 0}
      by simp
    show ⋀y. y ≤ nlength x ∧ ⌊x⌋N (Suc y) ∈ {□, 0} ⟹ nlength x ≤ y
      using False contents-def bit-symbols-canrepr
    by (metis diff-Suc-1 insert-iff le-neq-implies-less nat.simps(3) not-less-eq-eq numeral-3-eq-3 singletonD)
  qed
  then have i0: i0 = length ?zs
    using assms(1) by simp
  show ?thesis
  proof
    fix i
    consider i = 0 | Suc 0 ≤ i ∧ i < Suc (length ?zs) | i = Suc (length ?zs) | i > Suc (length ?zs)
      by linarith
    then have ⌊replicate (length ?zs) 0 @ [1]⌋ i =
        ((λi. if Suc 0 ≤ i ∧ i < Suc i0 then 0 else ⌊x⌋N i)(Suc i0 := 1)) i
        (is ?A i = ?B i)
    proof (cases)
      case 1
      then show ?thesis
        by (simp add: transplant-def)
    next
      case 2
      then have ?A i = 0
        by (metis One-nat-def Suc-le-lessD add.commute contents-def diff-Suc-1 length-Cons length-append
          length-replicate less-Suc-eq-0-disj less-imp-le-nat less-numeral-extra(3) list.size(3) nth-append
           nth-replicate plus-1-eq-Suc)
      moreover have ?B i = 0
        using i0 2 by simp
      ultimately show ?thesis
        by simp
    next
      case 3
      then show ?thesis
        using i0 canrepr-0 contents-inbounds nincr-canrepr nincr-zs nlength-0-simp nlength-Suc
        by (metis One-nat-def add.commute diff-Suc-1 fun-upd-apply length-Cons length-append
          length-replicate nth-append-length plus-1-eq-Suc zero-less-Suc)
    next
      case 4
      then show ?thesis
        using i0 by simp
    qed
    then show ⌊Suc x⌋N i = fst (constplant (⌊x⌋N, Suc 0) 0 i0 |:=| 1) i
      using nincr-zs l r by simp
  qed
 qed
qed
```

```
      ultimately show ?thesis
        by simp
    qed
    ultimately show ?thesis
      using assms(3) by simp
  qed
qed

lemma tm3:
  assumes i0 = (LEAST i. i ≤ nlength x ∧ ⌊x⌋N (Suc i) ∈ {□, 0})
    and ttt = 5 + 2 * i0
    and tps' = tps[j := (⌊Suc x⌋N, Suc 0)]
  shows transforms tm3 tps ttt tps'
  unfolding tm3-def
proof (tform tps: assms(1,3) time: assms(1,2))
  let ?tps = tps[j := (⌊Suc x⌋N, Suc (LEAST i. i ≤ nlength x ∧ ⌊x⌋N (Suc i) ∈ {□, 0}))]
  show clean-tape (?tps ! j)
    using clean-tape-ncontents by (simp add: assms(1,3))
qed

lemma tm3':
  assumes ttt = 5 + 2 * nlength x
    and tps' = tps[j := (⌊Suc x⌋N, Suc 0)]
  shows transforms tm3 tps ttt tps'
proof −
  let ?P = λi. i ≤ nlength x ∧ ⌊x⌋N (Suc i) ∈ {□, 0}
  define i0 where i0 = Least ?P
  have i0 ≤ nlength x ∧ ⌊x⌋N (Suc i0) ∈ {□, 0}
    using LeastI[of ?P nlength x] i0-def by simp
  then have 5 + 2 * i0 ≤ 5 + 2 * nlength x
    by simp
  moreover have transforms tm3 tps (5 + 2 * i0) tps'
    using assms tm3 i0-def by simp
  ultimately show ?thesis
    using transforms-monotone assms(1) by simp
qed

end

end

lemma transforms-tm-incrI [transforms-intros]:
  assumes j < k
    and length tps = k
    and tps ! j = (⌊x⌋N, 1)
    and ttt = 5 + 2 * nlength x
    and tps' = tps[j := (⌊Suc x⌋N, 1)]
  shows transforms (tm-incr j) tps ttt tps'
proof −
  interpret loc: turing-machine-incr j .
  show ?thesis
    using assms loc.tm3' loc.tm3-eq-tm-incr by simp
qed
```

Incrementing multiple times

Adding a constant by iteratively incrementing is not exactly efficient, but it still only takes constant time and thus does not endanger any time bounds.

```
fun tm-plus-const :: nat ⇒ tapeidx ⇒ machine where
  tm-plus-const 0 j = [] |
  tm-plus-const (Suc c) j = tm-plus-const c j ;; tm-incr j

lemma tm-plus-const-tm:
```

assumes $k \geq 2$ **and** $G \geq 4$ **and** $0 < j$ **and** $j < k$
shows *turing-machine k G* (*tm-plus-const c j*)
using *assms Nil-tm tm-incr-tm* **by** (*induction c*) *simp-all*

lemma *transforms-tm-plus-constI* [*transforms-intros*]:
fixes *c* :: *nat*
assumes $j < k$
and $j > 0$
and *length tps* = *k*
and *tps* ! $j = (\lfloor x \rfloor_N, 1)$
and *ttt* = $c * (5 + 2 *$ *nlength* $(x + c))$
and $tps' = tps[j := (\lfloor x + c \rfloor_N, 1)]$
shows *transforms* (*tm-plus-const c j*) *tps ttt tps'*
using *assms*(*5*,*6*,*4*)
proof (*induction c arbitrary*: *ttt tps'*)
case *0*
then show *?case*
using *transforms-Nil assms*
by (*metis add-cancel-left-right list-update-id mult-eq-0-iff tm-plus-const.simps*(*1*))
next
case (*Suc c*)
define *tpsA* **where** $tpsA = tps[j := (\lfloor x + c \rfloor_N, 1)]$
let $?ttt = c * (5 + 2 * nlength\ (x + c)) + (5 + 2 * nlength\ (x + c))$
have *transforms* (*tm-plus-const c j* ;; *tm-incr j*) *tps ?ttt tps'*
proof (*tform tps*: *assms*)
show *transforms* (*tm-plus-const c j*) *tps* $(c * (5 + 2 * nlength\ (x + c)))$ *tpsA*
using *tpsA-def assms Suc* **by** *simp*
show $j <$ *length tpsA*
using *tpsA-def assms*(*1*,*3*) **by** *simp*
show *tpsA* ! $j = (\lfloor x + c \rfloor_N, 1)$
using *tpsA-def assms*(*1*,*3*) **by** *simp*
show $tps' = tpsA[j := (\lfloor Suc\ (x + c) \rfloor_N, 1)]$
using *tpsA-def assms Suc* **by** (*metis add-Suc-right list-update-overwrite*)
qed
moreover have $?ttt \leq ttt$
proof −
have $?ttt = Suc\ c * (5 + 2 * nlength\ (x + c))$
by *simp*
also have ... $\leq Suc\ c * (5 + 2 * nlength\ (x + Suc\ c))$
using *nlength-mono Suc-mult-le-cancel1* **by** *auto*
finally show $?ttt \leq ttt$
using *Suc* **by** *simp*
qed
ultimately have *transforms* (*tm-plus-const c j* ;; *tm-incr j*) *tps ttt tps'*
using *transforms-monotone* **by** *simp*
then show *?case*
by *simp*
qed

2.7.3 Decrementing

Decrementing a number is almost like incrementing but with the symbols **0** and **1** swapped. One difference is that in order to get a canonical symbol sequence, a trailing zero must be removed, whereas incrementing cannot result in a trailing zero. Another difference is that decrementing the number zero yields zero.

The next function returns the leftmost symbol **1**, that is, the one that needs to be flipped.

definition *first1* :: *symbol list* ⇒ *nat* **where**
first1 zs ≡ *LEAST i*. $i <$ *length zs* ∧ *zs* ! $i =$ **1**

lemma *canonical-ex-3*:
assumes *canonical zs* **and** $zs \neq []$
shows $\exists\, i <$ *length zs*. *zs* ! $i =$ **1**
using *assms canonical-def* **by** (*metis One-nat-def Suc-pred last-conv-nth length-greater-0-conv lessI*)

lemma *canonical-first1*:
  assumes *canonical zs* **and** *zs* ≠ []
  shows *first1 zs* < *length zs* ∧ *zs* ! *first1 zs* = **1**
  using *assms canonical-ex-3* **by** (*metis* (*mono-tags*, *lifting*) *LeastI first1-def*)

lemma *canonical-first1-less*:
  assumes *canonical zs* **and** *zs* ≠ []
  shows ∀ *i*<*first1 zs*. *zs* ! *i* = **0**
proof −
  have ∀ *i*<*first1 zs*. *zs* ! *i* ≠ **1**
    using *assms first1-def canonical-first1 not-less-Least* **by** *fastforce*
  then show *?thesis*
    using *assms canonical-def* **by** (*meson canonical-first1 less-trans*)
qed

The next function describes how the canonical representation of the decremented symbol sequence looks like. It has special cases for the empty sequence and for sequences whose only **1** is the most significant digit.

definition *ndecr* :: *symbol list* ⇒ *symbol list* **where**
  ndecr zs ≡
    if zs = [] *then* []
    else if first1 zs = *length zs* − *1*
      then replicate (*first1 zs*) **1**
      else replicate (*first1 zs*) **1** @ [**0**] @ *drop* (*Suc* (*first1 zs*)) *zs*

lemma *canonical-ndecr*:
  assumes *canonical zs*
  shows *canonical* (*ndecr zs*)
proof −
  let *?i* = *first1 zs*
  consider
      zs = []
    | *zs* ≠ [] ∧ *first1 zs* = *length zs* − *1*
    | *zs* ≠ [] ∧ *first1 zs* < *length zs* − *1*
    using *canonical-first1 assms* **by** *fastforce*
  then show *?thesis*
  proof (*cases*)
    case *1*
    then show *?thesis*
      using *ndecr-def canonical-def* **by** *simp*
  next
    case *2*
    then show *?thesis*
      using *canonical-def ndecr-def not-less-eq* **by** *fastforce*
  next
    case *3*
    then have *Suc* (*first1 zs*) < *length zs*
      by *auto*
    then have *last* (*drop* (*Suc* (*first1 zs*)) *zs*) = **1**
      using *assms canonical-def 3* **by** *simp*
    moreover have *bit-symbols* (*replicate* (*first1 zs*) **1** @ [**0**] @ *drop* (*Suc* (*first1 zs*)) *zs*)
    proof −
      have *bit-symbols* (*replicate* (*first1 zs*) **1**)
        by *simp*
      moreover have *bit-symbols* [**0**]
        by *simp*
      moreover have *bit-symbols* (*drop* (*Suc* (*first1 zs*)) *zs*)
        using *assms canonical-def* **by** *simp*
      ultimately show *?thesis*
        using *bit-symbols-append* **by** *presburger*
    qed
    ultimately show *?thesis*

```
      using canonical-def ndecr-def 3 by auto
  qed
qed

lemma ndecr:
  assumes canonical zs
  shows num (ndecr zs) = num zs − 1
proof −
  let ?i = first1 zs
  consider zs = [] | zs ≠ [] ∧ first1 zs = length zs − 1 | zs ≠ [] ∧ first1 zs < length zs − 1
    using canonical-first1 assms by fastforce
  then show ?thesis
  proof (cases)
    case 1
    then show ?thesis
      using ndecr-def canrepr-0 canrepr by (metis zero-diff)
  next
    case 2
    then have less: zs ! i = 0 if i < first1 zs for i
      using that assms canonical-first1-less by simp
    have at: zs ! (first1 zs) = 1
      using 2 canonical-first1 assms by blast
    have zs = replicate (first1 zs) 0 @ [1] (is zs = ?zs)
    proof (rule nth-equalityI)
      show len: length zs = length ?zs
        using 2 by simp
      show zs ! i = ?zs ! i if i < length zs for i
      proof (cases i < first1 zs)
        case True
        then show ?thesis
          by (simp add: less nth-append)
      next
        case False
        then show ?thesis
          using len that at
          by (metis Suc-leI leD length-append-singleton length-replicate linorder-neqE-nat nth-append-length)
      qed
    qed
    moreover from this have ndecr zs = replicate (first1 zs) 3
      using ndecr-def 2 by simp
    ultimately show ?thesis
      using num-replicate2-eq-pow num-replicate3-eq-pow-minus-1 by metis
  next
    case 3
    then have less: zs ! i = 0 if i < ?i for i
      using that assms canonical-first1-less by simp
    have at: zs ! ?i = 1
      using 3 canonical-first1 assms by simp
    have zs: zs = replicate ?i 0 @ [1] @ drop (Suc ?i) zs (is zs = ?zs)
    proof (rule nth-equalityI)
      show len: length zs = length ?zs
        using 3 by auto
      show zs ! i = ?zs ! i if i < length zs for i
      proof −
        consider i < ?i | i = ?i | i > ?i
          by linarith
        then show ?thesis
        proof (cases)
          case 1
          then show ?thesis
            using less by (metis length-replicate nth-append nth-replicate)
        next
          case 2
```

then show *?thesis*
using *at* **by** (*metis append-Cons length-replicate nth-append-length*)
next
case *3*
have *?zs* = (*replicate ?i* **0** @ [**1**]) @ *drop* (*Suc ?i*) *zs*
by *simp*
then have *?zs* ! *i* = *drop* (*Suc ?i*) *zs* ! (*i* − *Suc ?i*)
using *3* **by** (*simp add*: *nth-append*)
then have *?zs* ! *i* = *zs* ! *i*
using *3 that* **by** *simp*
then show *?thesis*
by *simp*
qed
qed
qed
then have *ndecr zs* = *replicate ?i* **1** @ [**0**] @ *drop* (*Suc ?i*) *zs*
using *ndecr-def 3* **by** *simp*
then have *Suc* (*num* (*ndecr zs*)) = *Suc* (*num* ((*replicate ?i* **1** @ [**0**]) @ *drop* (*Suc ?i*) *zs*))
(**is** - = *Suc* (*num* (*?xs* @ *?ys*)))
by *simp*
also have ... = *Suc* (*num ?xs* + *2* ^ *length ?xs* ∗ *num ?ys*)
using *num-append* **by** *blast*
also have ... = *Suc* (*num ?xs* + *2* ^ *Suc ?i* ∗ *num ?ys*)
by *simp*
also have ... = *Suc* (*2* ^ *?i* − *1* + *2* ^ *Suc ?i* ∗ *num ?ys*)
using *num-replicate3-eq-pow-minus-1 num-trailing-zero*[*of 2 replicate ?i* **1**] **by** *simp*
also have ... = *2* ^ *?i* + *2* ^ *Suc ?i* ∗ *num ?ys*
by *simp*
also have ... = *num* (*replicate ?i* **0** @ [**1**]) + *2* ^ *Suc ?i* ∗ *num ?ys*
using *num-replicate2-eq-pow* **by** *simp*
also have ... = *num* ((*replicate ?i* **0** @ [**1**]) @ *?ys*)
using *num-append* **by** (*metis length-append-singleton length-replicate*)
also have ... = *num* (*replicate ?i* **0** @ [**1**] @ *?ys*)
by *simp*
also have ... = *num zs*
using *zs* **by** *simp*
finally have *Suc* (*num* (*ndecr zs*)) = *num zs* **.**
then show *?thesis*
by *simp*
qed
qed

The next Turing machine implements the function *ndecr*. It does nothing on the empty input, which represents zero. On other inputs it writes symbols **1** going right until it reaches a **1** symbol, which is guaranteed to happen for non-empty canonical representations. It then overwrites this **1** with **0**. If there is a blank symbol to the right of this **0**, the **0** is removed again.

definition *tm-decr* :: *tapeidx* ⇒ *machine* **where**
tm-decr j ≡
IF λ*rs*. *rs* ! *j* = □ *THEN*
[]
ELSE
tm-const-until j j {**1**} **1** ;;
tm-rtrans j (λ-. **0**) ;;
IF λ*rs*. *rs* ! *j* = □ *THEN*
tm-left j ;;
tm-write j □
ELSE
[]
ENDIF ;;
tm-cr j
ENDIF

lemma *tm-decr-tm*:

assumes $G \geq 4$ **and** $k \geq 2$ **and** $j < k$ **and** $0 < j$
shows *turing-machine k G* (*tm-decr j*)
unfolding *tm-decr-def*
using *assms tm-cr-tm tm-const-until-tm tm-rtrans-tm tm-left-tm tm-write-tm turing-machine-branch-turing-machine Nil-tm*
by *simp*

locale *turing-machine-decr* =
fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-const-until j j* {**1**} **1**
definition *tm2* ≡ *tm1* ;; *tm-rtrans j* (λ-. **0**)
definition *tm23* ≡ *tm-left j*
definition *tm24* ≡ *tm23* ;; *tm-write j* □
definition *tm25* ≡ *IF* λ*rs. rs* ! *j* = □ *THEN tm24 ELSE* [] *ENDIF*
definition *tm5* ≡ *tm2* ;; *tm25*
definition *tm6* ≡ *tm5* ;; *tm-cr j*
definition *tm7* ≡ *IF* λ*rs. rs* ! *j* = □ *THEN* [] *ELSE tm6 ENDIF*

lemma *tm7-eq-tm-decr*: *tm7* = *tm-decr j*
unfolding *tm1-def tm2-def tm23-def tm24-def tm25-def tm5-def tm6-def tm7-def tm-decr-def*
by *simp*

context
fixes *tps0* :: *tape list* **and** *xs* :: *symbol list* **and** *k* :: *nat*
assumes *jk*: *length tps0* = *k j* < *k*
and *can*: *canonical xs*
and *tps0*: *tps0* ! *j* = (⌊*xs*⌋, *1*)
begin

lemma *bs*: *bit-symbols xs*
using *can canonical-def* **by** *simp*

context
assumes *read-tps0*: *read tps0* ! *j* = □
begin

lemma *xs-Nil*: *xs* = []
using *tps0 jk tapes-at-read′ read-tps0 bs contents-inbounds*
by (*metis can canreprI canrepr-0 fst-conv ncontents-1-blank-iff-zero snd-conv*)

lemma *transforms-NilI*:
assumes *ttt* = *0*
and *tps′* = *tps0*[*j* := (⌊*ndecr xs*⌋, *1*)]
shows *transforms* [] *tps0 ttt tps′*
using *transforms-Nil xs-Nil ndecr-def tps0 assms* **by** (*metis Basics.transforms-Nil list-update-id*)

end

context
assumes *read-tps0′*: *read tps0* ! *j* ≠ □
begin

lemma *xs*: *xs* ≠ []
using *tps0 jk tapes-at-read′ read-tps0′ bs contents-inbounds*
by (*metis canrepr-0 fst-conv ncontents-1-blank-iff-zero snd-conv*)

lemma *first1*: *first1 xs* < *length xs xs* ! *first1 xs* = **1** ∀ *i*<*first1 xs*. *xs* ! *i* = **0**
using *canonical-first1*[*OF can xs*] *canonical-first1-less*[*OF can xs*] **by** *simp-all*

definition *tps1* ≡ *tps0*
[*j* := (⌊*replicate* (*first1 xs*) **1** @ [**1**] @ (*drop* (*Suc* (*first1 xs*)) *xs*)⌋, *Suc* (*first1 xs*))]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *Suc* (*first1 xs*)
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
proof (*tform tps*: *tps1-def jk time*: *assms*)
show *rneigh* (*tps0* ! *j*) {**1**} (*first1 xs*)
proof (*rule rneighI*)
show (*tps0* ::: *j*) (*tps0* :#: *j* + *first1 xs*) $\in$ {**1**}
using *first1*(*1*,*2*) *tps0 jk* **by** (*simp add*: *Suc-leI*)
show $\bigwedge n'$. n' < *first1 xs* $\Longrightarrow$ (*tps0* ::: *j*) (*tps0* :#: *j* + n') $\notin$ {**1**}
using *first1*(*3*) *tps0 jk* **by** (*simp add*: *contents-def*)
qed
show *tps1* = *tps0*
[*j* := *tps0* ! *j* |+| *first1 xs*,
j := *constplant* (*tps0* ! *j*) **1** (*first1 xs*)]
proof −
have *tps1* ! *j* = *constplant* (*tps0* ! *j*) *3* (*first1 xs*)
(**is** - = *?rhs*)
proof −
have *fst ?rhs* = (λ*i*. *if 1* $\le$ *i* $\wedge$ *i* < *1* + *first1 xs then* **1** *else* $\lfloor$*xs*$\rfloor$ *i*)
using *tps0 jk constplant* **by** *auto*
also have ... = $\lfloor$*replicate* (*first1 xs*) **1** @ [**1**] @ *drop* (*Suc* (*first1 xs*)) *xs*$\rfloor$
proof
fix *i*
consider
i = *0*
| *i* $\ge$ *1* $\wedge$ *i* < *1* + *first1 xs*
| *i* = *1* + *first1 xs*
| *1* + *first1 xs* < *i* $\wedge$ *i* $\le$ *length xs*
| *i* > *length xs*
by *linarith*
then show (*if 1* $\le$ *i* $\wedge$ *i* < *1* + *first1 xs then* **1** *else* $\lfloor$*xs*$\rfloor$ *i*) =
$\lfloor$*replicate* (*first1 xs*) **1** @ [**1**] @ *drop* (*Suc* (*first1 xs*)) *xs*$\rfloor$ *i*
(**is** *?l* = *?r*)
proof (*cases*)
case *1*
then show *?thesis*
by *simp*
next
case *2*
then show *?thesis*
by (*smt* (*verit*) *One-nat-def Suc-diff-Suc add-diff-inverse-nat contents-inbounds first1*(*1*) *length-append length-drop length-replicate less-imp-le-nat less-le-trans list.size*(*3*) *list.size*(*4*) *not-le not-less-eq nth-append nth-replicate plus-1-eq-Suc*)
next
case *3*
then show *?thesis*
using *first1*
by (*smt* (*verit*) *One-nat-def Suc-diff-Suc Suc-leI add-diff-inverse-nat append-Cons contents-inbounds diff-Suc-1 length-append length-drop length-replicate less-SucI less-Suc-eq-0-disj list.size*(*3*) *list.size*(*4*) *not-less-eq nth-append-length*)
next
case *4*
then have *?r* = (*replicate* (*first1 xs*) **1** @ [**1**] @ *drop* (*Suc* (*first1 xs*)) *xs*) ! (*i* − *1*)
by *auto*
also have ... = ((*replicate* (*first1 xs*) **1** @ [**1**]) @ *drop* (*Suc* (*first1 xs*)) *xs*) ! (*i* − *1*)
by *simp*
also have ... = (*drop* (*Suc* (*first1 xs*)) *xs*) ! (*i* − *1* − *Suc* (*first1 xs*))
using *4*
by (*metis Suc-leI add-diff-inverse-nat gr-implies-not0 leD length-append-singleton length-replicate less-one nth-append plus-1-eq-Suc*)
also have ... = *xs* ! (*i* − *1*)

using *4* **by** (*metis Suc-leI add-diff-inverse-nat first1*(*1*) *gr-implies-not0 leD less-one nth-drop plus-1-eq-Suc*)
also have ... = ⌊*xs*⌋ *i*
using *4* **by** *simp*
also have ... = *?l*
using *4* **by** *simp*
finally have *?r* = *?l* **.**
then show *?thesis*
by *simp*
next
case *5*
then show *?thesis*
using *first1*(*1*) **by** *simp*
qed
qed
also have ... = *tps1* ::: *j*
using *tps1-def jk* **by** *simp*
finally have *fst ?rhs* = *fst* (*tps1* ! *j*) **.**
then show *?thesis*
using *tps1-def jk constplant tps0* **by** *simp*
qed
then show *?thesis*
using *tps1-def tps0 jk* **by** *simp*
qed
qed

definition *tps2* ≡ *tps0*
[*j* := (⌊*replicate* (*first1 xs*) **1** @ [**0**] @ *drop* (*Suc* (*first1 xs*)) *xs*⌋, *Suc* (*Suc* (*first1 xs*)))]

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *first1 xs* + *2*
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def*
proof (*tform tps*: *tps2-def tps1-def jk time*: *assms*)
show *tps2* = *tps1*[*j* := *tps1* ! *j* |:=| **0** |+| *1*]
using *tps1-def tps2-def jk contents-append-update* **by** *simp*
qed

definition *tps5* ≡ *tps0*
[*j* := (⌊*ndecr xs*⌋, *if read tps2* ! *j* = □ *then Suc* (*first1 xs*) *else Suc* (*Suc* (*first1 xs*)))]

context
assumes *read-tps2*: *read tps2* ! *j* = □
begin

lemma *proper-contents-outofbounds*:
assumes *proper-symbols zs* **and** ⌊*zs*⌋ *i* = □
shows *i* > *length zs*
using *contents-def proper-symbols-ne0 assms*
by (*metis Suc-diff-1 bot-nat-0.not-eq-extremum linorder-le-less-linear not-less-eq zero-neq-one*)

lemma *first1-eq*: *first1 xs* = *length xs* − *1*
proof −
have *tps2* ! *j* = (⌊*replicate* (*first1 xs*) **1** @ [**0**] @ *drop* (*Suc* (*first1 xs*)) *xs*⌋, *Suc* (*Suc* (*first1 xs*)))
(**is** - = (⌊*?zs*⌋, *?i*))
using *tps2-def jk* **by** *simp*
have *proper-symbols xs*
using *can bs* **by** *fastforce*
then have ∗: *proper-symbols ?zs*
using *proper-symbols-append*[*of* [**0**] *drop* (*Suc* (*first1 xs*)) *xs*] *proper-symbols-append*
by *simp*
have *read tps2* ! *j* = ⌊*?zs*⌋ *?i*
using *tps2-def jk tapes-at-read'*[*of j tps2*] **by** *simp*

then have $\lfloor ?zs \rfloor\ ?i = \Box$
using *read-tps2* **by** *simp*
then have $?i > length\ ?zs$
using $*$ *proper-contents-outofbounds* **by** *blast*
moreover have $length\ ?zs = length\ xs$
using *first1* **by** *simp*
ultimately have $Suc\ (first1\ xs) \geq length\ xs$
by *simp*
moreover have $length\ xs > 0$
using *xs* **by** *simp*
ultimately have $first1\ xs \geq length\ xs - 1$
by *simp*
then show *?thesis*
using *first1(1)* **by** *simp*
qed

lemma *drop-xs-Nil*: $drop\ (Suc\ (first1\ xs))\ xs = []$
using *first1-eq xs* **by** *simp*

lemma *tps2-eq*: $tps2 = tps0[j := (\lfloor replicate\ (first1\ xs)\ \mathbf{1}\ @\ [\mathbf{0}] \rfloor,\ Suc\ (Suc\ (first1\ xs)))]$
using *tps2-def drop-xs-Nil jk* **by** *simp*

definition $tps23 \equiv tps0$
$[j := (\lfloor replicate\ (first1\ xs)\ \mathbf{1}\ @\ [\mathbf{0}] \rfloor,\ Suc\ (first1\ xs))]$

lemma *tm23* [*transforms-intros*]:
assumes $ttt = 1$
shows *transforms tm23 tps2 ttt tps23*
unfolding *tm23-def*
proof (*tform tps*: *tps2-def tps23-def jk time*: *assms*)
show $tps23 = tps2[j := tps2\ !\ j\ |{-}|\ 1]$
using *tps23-def tps2-eq jk* **by** *simp*
qed

definition $tps24 \equiv tps0$
$[j := (\lfloor replicate\ (first1\ xs)\ \mathbf{1} \rfloor,\ Suc\ (first1\ xs))]$

lemma *tm24*:
assumes $ttt = 2$
shows *transforms tm24 tps2 ttt tps24*
unfolding *tm24-def*
proof (*tform tps*: *tps23-def tps24-def time*: *assms*)
show $tps24 = tps23[j := tps23\ !\ j\ |{:=}|\ \Box]$
proof $-$
have $tps23\ !\ j\ |{:=}|\ \Box = (\lfloor replicate\ (first1\ xs)\ \mathbf{1}\ @\ [\mathbf{0}] \rfloor,\ Suc\ (first1\ xs))\ |{:=}|\ \Box$
using *tps23-def jk* **by** *simp*
then have $tps23\ !\ j\ |{:=}|\ \Box = (\lfloor replicate\ (first1\ xs)\ \mathbf{1}\ @\ [\Box] \rfloor,\ Suc\ (first1\ xs))$
using *contents-append-update* **by** *auto*
then have $tps23\ !\ j\ |{:=}|\ \Box = (\lfloor replicate\ (first1\ xs)\ \mathbf{1} \rfloor,\ Suc\ (first1\ xs))$
using *contents-append-blanks* **by** (*metis replicate-0 replicate-Suc*)
moreover have $tps24\ !\ j = (\lfloor replicate\ (first1\ xs)\ \mathbf{1} \rfloor,\ Suc\ (first1\ xs))$
using *tps24-def jk* **by** *simp*
ultimately show *?thesis*
using *tps23-def tps24-def* **by** *auto*
qed
qed

corollary *tm24'* [*transforms-intros*]:
assumes $ttt = 2$ **and** $tps' = tps0[j := (\lfloor ndecr\ xs \rfloor,\ Suc\ (first1\ xs))]$
shows *transforms tm24 tps2 ttt tps'*
proof $-$
have $tps24 = tps0[j := (\lfloor ndecr\ xs \rfloor,\ Suc\ (first1\ xs))]$
using *tps24-def jk ndecr-def first1-eq xs* **by** *simp*

```
  then show ?thesis
    using assms tm24 by simp
qed

end

context
  assumes read-tps2': read tps2 ! j ≠ □
begin

lemma first1-neq: first1 xs ≠ length xs − 1
proof (rule ccontr)
  assume eq: ¬ first1 xs ≠ length xs − 1

  have tps2 ! j = (⌊replicate (first1 xs) 1 @ [0] @ drop (Suc (first1 xs)) xs⌋, Suc (Suc (first1 xs)))
      (is - = (⌊?zs⌋, ?i))
    using tps2-def jk by simp
  have length ?zs = length xs
    using first1 by simp
  then have Suc (Suc (first1 xs)) = Suc (length ?zs)
    using xs eq by simp
  then have ∗: ⌊?zs⌋ ?i = 0
    using contents-outofbounds by simp

  have read tps2 ! j = ⌊?zs⌋ ?i
    using tps2-def jk tapes-at-read'[of j tps2] by simp
  then have ⌊?zs⌋ ?i ≠ □
    using read-tps2' by simp
  then show False
    using ∗ by simp
qed

lemma tps2: tps2 = tps0[j := (⌊ndecr xs⌋, Suc (Suc (first1 xs)))]
  using tps2-def ndecr-def first1-neq xs by simp

end

lemma tm25 [transforms-intros]:
  assumes ttt = (if read tps2 ! j = □ then 4 else 1)
  shows transforms tm25 tps2 ttt tps5
  unfolding tm25-def by (tform tps: tps2 tps5-def time: assms)

lemma tm5 [transforms-intros]:
  assumes ttt = first1 xs + 2 + (if read tps2 ! j = □ then 4 else 1)
  shows transforms tm5 tps0 ttt tps5
  unfolding tm5-def by (tform time: assms)

definition tps6 ≡ tps0
  [j := (⌊ndecr xs⌋, 1)]

lemma tm6:
  assumes ttt = first1 xs + 2 + (if read tps2 ! j = □ then 4 else 1) + (tps5 :#: j + 2)
  shows transforms tm6 tps0 ttt tps6
  unfolding tm6-def
proof (tform tps: tps5-def tps6-def jk time: assms)
  show clean-tape (tps5 ! j)
  proof −
    have tps5 ::: j = ⌊ndecr xs⌋
      using tps5-def jk by simp
    moreover have bit-symbols (ndecr xs)
      using canonical-ndecr can canonical-def by simp
    ultimately show ?thesis
      using One-nat-def Suc-1 Suc-le-lessD clean-contents-proper
```

```
      by (metis contents-clean-tape' lessI one-less-numeral-iff semiring-norm(77))
  qed
qed

lemma tm6' [transforms-intros]:
  assumes ttt = 2 * first1 xs + 9
  shows transforms tm6 tps0 ttt tps6
proof -
  let ?ttt = first1 xs + 2 + (if read tps2 ! j = □ then 4 else 1) + (tps5 :#: j + 2)
  have tps5 :#: j = (if read tps2 ! j = □ then Suc (first1 xs) else Suc (Suc (first1 xs)))
    using tps5-def jk by simp
  then have ?ttt ≤ ttt
    using assms by simp
  then show ?thesis
    using tm6 transforms-monotone assms by simp
qed

end

definition tps7 ≡ tps0[j := (⌊ndecr xs⌋, 1)]

lemma tm7:
  assumes ttt = 8 + 2 * length xs
  shows transforms tm7 tps0 ttt tps7
  unfolding tm7-def
proof (tform tps: tps6-def tps7-def time: assms)
  show tps7 = tps0 if read tps0 ! j = □
    using that ndecr-def tps0 tps7-def xs-Nil jk by (simp add: list-update-same-conv)
  show 2 * first1 xs + 9 + 1 ≤ ttt if read tps0 ! j ≠ □
  proof -
    have length xs > 0
      using that xs by simp
    then show ?thesis
      using first1(1) that assms by simp
  qed
qed

end

end

lemma transforms-tm-decrI [transforms-intros]:
  fixes tps tps' :: tape list and n :: nat and k ttt :: nat
  assumes j < k length tps = k
  assumes tps ! j = (⌊n⌋_N, 1)
  assumes ttt = 8 + 2 * nlength n
  assumes tps' = tps[j := (⌊n - 1⌋_N, 1)]
  shows transforms (tm-decr j) tps ttt tps'
proof -
  let ?xs = canrepr n
  have can: canonical ?xs
    using canonical-canrepr by simp
  have tps0: tps ! j = (⌊?xs⌋, 1)
    using assms by simp
  have tps': tps' = tps[j := (⌊ndecr ?xs⌋, 1)]
    using ndecr assms(5) by (metis canrepr canreprI can canonical-ndecr)
  interpret loc: turing-machine-decr j .
  have transforms loc.tm7 tps ttt tps'
    using loc.tm7 loc.tps7-def by (metis assms(1,2,4) can tps' tps0)
  then show ?thesis
    using loc.tm7-eq-tm-decr by simp
qed
```

2.7.4 Addition

In this section we construct a Turing machine that adds two numbers in canonical representation each given on a separate tape and overwrites the second number with the sum. The TM implements the common algorithm with carry starting from the least significant digit.

Given two symbol sequences *xs* and *ys* representing numbers, the next function computes the carry bit that occurs in the i-th position. For the least significant position, 0, there is no carry (that is, it is 0); for position $i + 1$ the carry is the sum of the bits of *xs* and *ys* in position i and the carry for position i. The function gives the carry as symbol **0** or **1**, except for position 0, where it is the start symbol ▷. The start symbol represents the same bit as the symbol **0** as defined by *todigit*. The reason for this special treatment is that the TM will store the carry on a memorization tape (see Section 2.5), which initially contains the start symbol.

```
fun carry :: symbol list ⇒ symbol list ⇒ nat ⇒ symbol where
  carry xs ys 0 = 1 |
  carry xs ys (Suc i) = tosym ((todigit (digit xs i) + todigit (digit ys i) + todigit (carry xs ys i)) div 2)
```

The next function specifies the i-th digit of the sum.

```
definition sumdigit :: symbol list ⇒ symbol list ⇒ nat ⇒ symbol where
  sumdigit xs ys i ≡ tosym ((todigit (digit xs i) + todigit (digit ys i) + todigit (carry xs ys i)) mod 2)

lemma carry-sumdigit: todigit (sumdigit xs ys i) + 2 ∗ (todigit (carry xs ys (Suc i))) =
    todigit (carry xs ys i) + todigit (digit xs i) + todigit (digit ys i)
  using sumdigit-def by simp

lemma carry-sumdigit-eq-sum:
  num xs + num ys =
   num (map (sumdigit xs ys) [0..<t]) + 2 ^ t ∗ todigit (carry xs ys t) + 2 ^ t ∗ num (drop t xs) + 2 ^ t ∗ num
(drop t ys)
proof (induction t)
  case 0
  then show ?case
    using num-def by simp
next
  case (Suc t)
  let ?z = sumdigit xs ys
  let ?c = carry xs ys
  let ?zzz = map ?z [0..<Suc t]
  have num (take (Suc t) ?zzz) = num (take t ?zzz) + 2 ^ t ∗ todigit (digit ?zzz t)
    using num-take-Suc by blast
  moreover have take (Suc t) ?zzz = map (sumdigit xs ys) [0..<Suc t]
    by simp
  moreover have take t ?zzz = map (sumdigit xs ys) [0..<t]
    by simp
  ultimately have 1: num (map ?z [0..<Suc t]) = num (map ?z [0..<t]) + 2 ^ t ∗ todigit (digit ?zzz t)
    by simp

  have 2: digit ?zzz t = sumdigit xs ys t
    using digit-def
    by (metis One-nat-def add-Suc diff-add-inverse length-map length-upt lessI nth-map-upt plus-1-eq-Suc)

  have todigit (?z t) + 2 ∗ (todigit (carry xs ys (Suc t))) =
      todigit (carry xs ys t) + todigit (digit xs t) + todigit (digit ys t)
    using carry-sumdigit .
  then have 2 ^ t ∗ (todigit (?z t) + 2 ∗ (todigit (?c (Suc t)))) =
      2 ^ t ∗ (todigit (?c t) + todigit (digit xs t) + todigit (digit ys t))
    by simp
  then have 2 ^ t ∗ todigit (?z t) + 2 ^ t ∗ 2 ∗ todigit (?c (Suc t)) =
      2 ^ t ∗ todigit (?c t) + 2 ^ t ∗ todigit (digit xs t) + 2 ^ t ∗ todigit (digit ys t)
    using add-mult-distrib2 by simp
  then have num (map ?z [0..<t]) + 2 ^ t ∗ (todigit (?z t)) + 2 ^ Suc t ∗ (todigit (?c (Suc t))) =
      num (map ?z [0..<t]) + 2 ^ t ∗ (todigit (?c t)) + 2 ^ t ∗ (todigit (digit xs t)) + 2^t ∗ (todigit (digit ys t))
    by simp
```

then have *num* (*map ?z* [*0*..<*Suc t*]) + *2* ^ *Suc t* ∗ (*todigit* (*?c* (*Suc t*))) =
num (*map ?z* [*0*..<*t*]) + *2* ^ *t* ∗ *todigit* (*?c t*) + *2* ^ *t* ∗ *todigit* (*digit xs t*) + *2* ^ *t* ∗ *todigit* (*digit ys t*)
using *1 2* **by** *simp*
then have *num* (*map ?z* [*0*..<*Suc t*]) + *2* ^ *Suc t* ∗ (*todigit* (*?c* (*Suc t*))) +
2 ^ *Suc t* ∗ *num* (*drop* (*Suc t*) *xs*) + *2* ^ *Suc t* ∗ *num* (*drop* (*Suc t*) *ys*) =
num (*map ?z* [*0*..<*t*]) + *2* ^ *t* ∗ *todigit* (*?c t*) + *2* ^ *t* ∗ *todigit* (*digit xs t*) + *2* ^ *t* ∗ *todigit* (*digit ys t*) +
2 ^ *Suc t* ∗ *num* (*drop* (*Suc t*) *xs*) + *2* ^ *Suc t* ∗ *num* (*drop* (*Suc t*) *ys*)
by *simp*
also have ... = *num* (*map ?z* [*0*..<*t*]) + *2* ^ *t* ∗ (*todigit* (*?c t*)) +
2 ^ *t* ∗ (*todigit* (*digit xs t*) + *2* ∗ *num* (*drop* (*Suc t*) *xs*)) + *2* ^ *t* ∗ (*todigit* (*digit ys t*) + *2* ∗ *num* (*drop* (*Suc t*) *ys*))
by (*simp add*: *add-mult-distrib2*)
also have ... = *num* (*map ?z* [*0*..<*t*]) + *2* ^ *t* ∗ (*todigit* (*?c t*)) +
2 ^ *t* ∗ *num* (*drop t xs*) + *2* ^ *t* ∗ *num* (*drop t ys*)
using *num-drop* **by** *metis*
also have ... = *num xs* + *num ys*
using *Suc* **by** *simp*
finally show *?case*
by *simp*
qed

lemma *carry-le*:
assumes *symbols-lt 4 xs* **and** *symbols-lt 4 ys*
shows *carry xs ys t* ≤ **1**
proof (*induction t*)
case *0*
then show *?case*
by *simp*
next
case (*Suc t*)
then have *todigit* (*carry xs ys t*) ≤ *1*
by *simp*
moreover have *todigit* (*digit xs t*) ≤ *1*
using *assms*(*1*) *digit-def* **by** *auto*
moreover have *todigit* (*digit ys t*) ≤ *1*
using *assms*(*2*) *digit-def* **by** *auto*
ultimately show *?case*
by *simp*
qed

lemma *num-sumdigit-eq-sum*:
assumes *length xs* ≤ *n*
and *length ys* ≤ *n*
and *symbols-lt 4 xs*
and *symbols-lt 4 ys*
shows *num xs* + *num ys* = *num* (*map* (*sumdigit xs ys*) [*0*..<*Suc n*])
proof −
have *num xs* + *num ys* =
num (*map* (*sumdigit xs ys*) [*0*..<*Suc n*]) + *2* ^ *Suc n* ∗ *todigit* (*carry xs ys* (*Suc n*)) +
2 ^ *Suc n* ∗ *num* (*drop* (*Suc n*) *xs*) + *2* ^ *Suc n* ∗ *num* (*drop* (*Suc n*) *ys*)
using *carry-sumdigit-eq-sum* **by** *blast*
also have ... = *num* (*map* (*sumdigit xs ys*) [*0*..<*Suc n*]) + *2* ^ *Suc n* ∗ *todigit* (*carry xs ys* (*Suc n*))
using *assms*(*1*,*2*) **by** (*simp add*: *num-def*)
also have ... = *num* (*map* (*sumdigit xs ys*) [*0*..<*Suc n*])
proof −
have *digit xs n* = *0*
using *assms*(*1*) *digit-def* **by** *simp*
moreover have *digit ys n* = *0*
using *assms*(*2*) *digit-def* **by** *simp*
ultimately have (*digit xs n* + *digit ys n* + *todigit* (*carry xs ys n*)) *div 2* = *0*
using *carry-le*[*OF assms*(*3*,*4*), *of n*] **by** *simp*
then show *?thesis*
by *auto*

  qed
  finally show *?thesis* **.**
qed

lemma *num-sumdigit-eq-sum′*:
  assumes *symbols-lt 4 xs* **and** *symbols-lt 4 ys*
  shows *num xs* + *num ys* = *num* (*map* (*sumdigit xs ys*) [*0*..<*Suc* (*max* (*length xs*) (*length ys*))])
  using *assms num-sumdigit-eq-sum* **by** *simp*

lemma *num-sumdigit-eq-sum′′*:
  assumes *bit-symbols xs* **and** *bit-symbols ys*
  shows *num xs* + *num ys* = *num* (*map* (*sumdigit xs ys*) [*0*..<*Suc* (*max* (*length xs*) (*length ys*))])
proof −
  have *symbols-lt 4 xs*
    using *assms*(*1*) **by** *auto*
  moreover have *symbols-lt 4 ys*
    using *assms*(*2*) **by** *auto*
  ultimately show *?thesis*
    using *num-sumdigit-eq-sum′* **by** *simp*
qed

lemma *sumdigit-bit-symbols*: *bit-symbols* (*map* (*sumdigit xs ys*) [*0*..<*t*])
  using *sumdigit-def* **by** *auto*

The core of the addition Turing machine is the following command. It scans the symbols on tape j_1 and j_2 in lockstep until it reaches blanks on both tapes. In every step it adds the symbols on both tapes and the symbol on the last tape, which is a memorization tape storing the carry bit. The sum of these three bits modulo 2 is written to tape j_2 and the new carry to the memorization tape.

definition *cmd-plus* :: *tapeidx* ⇒ *tapeidx* ⇒ *command* **where**
  cmd-plus j1 j2 rs ≡
    (*if rs* ! *j1* = □ ∧ *rs* ! *j2* = □ *then 1 else 0*,
     (*map* (λ*j*.
       if j = *j1 then* (*rs* ! *j*, *Right*)
       else if j = *j2 then* (*tosym* ((*todigit* (*rs* ! *j1*) + *todigit* (*rs* ! *j2*) + *todigit* (*last rs*)) *mod 2*), *Right*)
       else if j = *length rs* − *1 then* (*tosym* ((*todigit* (*rs* ! *j1*) + *todigit* (*rs* ! *j2*) + *todigit* (*last rs*)) *div 2*), *Stay*)
       else (*rs* ! *j*, *Stay*)) [*0*..<*length rs*]))

lemma *sem-cmd-plus*:
  assumes *j1* ≠ *j2*
    and *j1* < *k* − *1*
    and *j2* < *k* − *1*
    and *j2* > *0*
    and *length tps* = *k*
    and *bit-symbols xs*
    and *bit-symbols ys*
    and *tps* ! *j1* = (⌊*xs*⌋, *Suc t*)
    and *tps* ! *j2* = (⌊*map* (*sumdigit xs ys*) [*0*..<*t*] @ *drop t ys*⌋, *Suc t*)
    and *last tps* = ⌈*carry xs ys t*⌉
    and *rs* = *read tps*
    and *tps′* = *tps*
      [*j1* := *tps*!*j1* |+| *1*,
       j2 := *tps*!*j2* |:=| *sumdigit xs ys t* |+| *1*,
       length tps − *1* := ⌈*carry xs ys* (*Suc t*)⌉]
  shows *sem* (*cmd-plus j1 j2*) (*0*, *tps*) = (*if t* < *max* (*length xs*) (*length ys*) *then 0 else 1*, *tps′*)
proof
  have *k* ≥ *2*
    using *assms*(*3*,*4*) **by** *simp*
  have *rs1*: *rs* ! *j1* = *digit xs t*
    using *assms*(*2*,*5*,*8*,*11*) *digit-def read-def contents-def* **by** *simp*
  let *?zs* = *map* (*sumdigit xs ys*) [*0*..<*t*] @ *drop t ys*
  have *rs2*: *rs* ! *j2* = *digit ys t*
  proof (*cases t* < *length ys*)
    case *True*

```
  then have ?zs ! t = ys ! t
    by (simp add: nth-append)
  then show ?thesis
    using assms(3,5,9,11) digit-def read-def contents-def by simp
next
  case False
  then have length ?zs = t
    by simp
  then have ⌊?zs⌋ (Suc t) = □
    using False contents-def by simp
  then show ?thesis
    using digit-def read-def contents-def False assms(3,5,9,11) by simp
qed
have rs3: last rs = carry xs ys t
  using ‹k ≥ 2› assms onesie-read onesie-def read-def read-length tapes-at-read'
 by (metis (no-types, lifting) diff-less last-conv-nth length-greater-0-conv less-one list.size(3) not-numeral-le-zero)
have ∗: tosym ((todigit (rs ! j1) + todigit (rs ! j2) + todigit (last rs)) mod 2) = sumdigit xs ys t
  using rs1 rs2 rs3 sumdigit-def by simp

have ¬ (digit xs t = 0 ∧ digit ys t = 0) if t < max (length xs) (length ys)
  using assms(6,7) digit-def that by auto
then have 4: ¬ (rs ! j1 = 0 ∧ rs ! j2 = 0) if t < max (length xs) (length ys)
  using rs1 rs2 that by simp
then have fst1: fst (sem (cmd-plus j1 j2) (0, tps)) = fst (0, tps') if t < max (length xs) (length ys)
  using that cmd-plus-def assms(11) by (smt (verit, ccfv-threshold) fst-conv prod.sel(2) sem)

have digit xs t = 0 ∧ digit ys t = 0 if t ≥ max (length xs) (length ys)
  using that digit-def by simp
then have 5: rs ! j1 = □ ∧ rs ! j2 = □ if t ≥ max (length xs) (length ys)
  using rs1 rs2 that by simp
then have fst (sem (cmd-plus j1 j2) (0, tps)) = fst (1, tps') if t ≥ max (length xs) (length ys)
  using that cmd-plus-def assms(11) by (smt (verit, ccfv-threshold) fst-conv prod.sel(2) sem)
then show fst (sem (cmd-plus j1 j2) (0, tps)) = fst (if t < max (length xs) (length ys) then 0 else 1, tps')
  using fst1 by (simp add: not-less)

show snd (sem (cmd-plus j1 j2) (0, tps)) = snd (if t < max (length xs) (length ys) then 0 else 1, tps')
proof (rule snd-semI)
  show proper-command k (cmd-plus j1 j2)
    using cmd-plus-def by simp
  show length tps = k
    using assms(5) .
  show length tps' = k
    using assms(5,12) by simp
  have len: length (read tps) = k
    by (simp add: assms read-length)
  show act (cmd-plus j1 j2 (read tps) [!] j) (tps ! j) = tps' ! j
    if j < k for j
  proof −
    have j: j < length tps
      using len that assms(5) by simp
    consider
        j = j1
      | j ≠ j1 ∧ j = j2
      | j ≠ j1 ∧ j ≠ j2 ∧ j = length rs − 1
      | j ≠ j1 ∧ j ≠ j2 ∧ j ≠ length rs − 1
      by auto
    then show ?thesis
    proof (cases)
      case 1
      then have cmd-plus j1 j2 (read tps) [!] j = (read tps ! j, Right)
        using that len cmd-plus-def by simp
      then have act (cmd-plus j1 j2 (read tps) [!] j) (tps ! j) = tps ! j |+| 1
        using act-Right[OF j] by simp
```

```
      moreover have tps' ! j = tps ! j |+| 1
        using assms(1,2,5,12) that 1 by simp
      ultimately show ?thesis
        by simp
    next
      case 2
      then have cmd-plus j1 j2 (read tps) [!] j =
          (tosym ((todigit (rs ! j1) + todigit (rs ! j2) + todigit (last rs)) mod 2), Right)
        using that len cmd-plus-def assms(11) by simp
      then have cmd-plus j1 j2 (read tps) [!] j = (sumdigit xs ys t, Right)
        using ∗ by simp
      moreover have tps' ! j2 = tps!j2 |:=| sumdigit xs ys t |+| 1
        using assms(3,5,12) by simp
      ultimately show ?thesis
        using act-Right' 2 by simp
    next
      case 3
      then have cmd-plus j1 j2 (read tps) [!] j =
          (tosym ((todigit (rs ! j1) + todigit (rs ! j2) + todigit (last rs)) div 2), Stay)
        using that len cmd-plus-def assms(11) by simp
      then have cmd-plus j1 j2 (read tps) [!] j = (carry xs ys (Suc t), Stay)
        using rs1 rs2 rs3 by simp
      moreover have tps' ! (length tps − 1) = ⌈carry xs ys (Suc t)⌉
        using 3 assms(5,11,12) len that by simp
      ultimately show ?thesis
        using 3 act-onesie assms(3,5,10,11) len
        by (metis add-diff-inverse-nat last-length less-nat-zero-code nat-diff-split-asm plus-1-eq-Suc)
    next
      case 4
      then have cmd-plus j1 j2 (read tps) [!] j = (read tps ! j, Stay)
        using that len cmd-plus-def assms(11) by simp
      then have act (cmd-plus j1 j2 (read tps) [!] j) (tps ! j) = tps ! j
        using act-Stay[OF j] by simp
      moreover have tps' ! j = tps ! j
        using that 4 len assms(5,11,12) by simp
      ultimately show ?thesis
        by simp
    qed
  qed
 qed
qed

lemma contents-map-append-drop:
 ⌊map f [0..<t] @ drop t zs⌋(Suc t := f t) = ⌊map f [0..<Suc t] @ drop (Suc t) zs⌋
proof (cases t < length zs)
 case lt: True
 then have t-lt: t < length (map f [0..<t] @ drop t zs)
  by simp
 show ?thesis
 proof
  fix x
  consider
     x = 0
   | x > 0 ∧ x < Suc t
   | x = Suc t
   | x > Suc t ∧ x ≤ length zs
   | x > Suc t ∧ x > length zs
   by linarith
  then show (⌊map f [0..<t] @ drop t zs⌋(Suc t := f t)) x =
     ⌊map f [0..<Suc t] @ drop (Suc t) zs⌋ x
     (is ?lhs x = ?rhs x)
  proof (cases)
   case 1
```

then show *?thesis*
using *contents-def* **by** *simp*
next
case *2*
then have *?lhs* $x = (map\ f\ [0..<t]\ @\ drop\ t\ zs)\ !\ (x - 1)$
using *contents-def* **by** *simp*
moreover have $x - 1 < t$
using *2* **by** *auto*
ultimately have *left*: *?lhs* $x = f\ (x - 1)$
by (*metis add.left-neutral diff-zero length-map length-upt nth-append nth-map-upt*)
have *?rhs* $x = (map\ f\ [0..<Suc\ t]\ @\ drop\ (Suc\ t)\ zs)\ !\ (x - 1)$
using *2 contents-def* **by** *simp*
moreover have $x - 1 < Suc\ t$
using *2* **by** *auto*
ultimately have *?rhs* $x = f\ (x - 1)$
by (*metis diff-add-inverse diff-zero length-map length-upt nth-append nth-map-upt*)
then show *?thesis*
using *left* **by** *simp*
next
case *3*
then show *?thesis*
using *contents-def lt*
by (*smt* (*verit*, *ccfv-threshold*) *One-nat-def Suc-leI add-Suc append-take-drop-id diff-Suc-1 diff-zero fun-upd-same length-append length-map length-take length-upt lessI min-absorb2 nat.simps*(*3*) *nth-append nth-map-upt plus-1-eq-Suc*)
next
case *4*
then have *?lhs* $x = \lfloor map\ f\ [0..<t]\ @\ drop\ t\ zs \rfloor\ x$
using *contents-def* **by** *simp*
then have *?lhs* $x = (map\ f\ [0..<t]\ @\ drop\ t\ zs)\ !\ (x - 1)$
using *4 contents-def* **by** *simp*
then have *left*: *?lhs* $x = drop\ t\ zs\ !\ (x - 1 - t)$
using *4*
by (*metis Suc-lessE diff-Suc-1 length-map length-upt less-Suc-eq-le less-or-eq-imp-le minus-nat.diff-0 not-less-eq nth-append*)
have $x \leq length\ (map\ f\ [0..<Suc\ t]\ @\ drop\ (Suc\ t)\ zs)$
using *4 lt* **by** *auto*
moreover have $x > 0$
using *4* **by** *simp*
ultimately have *?rhs* $x = (map\ f\ [0..<Suc\ t]\ @\ drop\ (Suc\ t)\ zs)\ !\ (x - 1)$
using *4 contents-inbounds* **by** *simp*
moreover have $x - 1 \geq Suc\ t$
using *4* **by** *auto*
ultimately have *?rhs* $x = drop\ (Suc\ t)\ zs\ !\ (x - 1 - Suc\ t)$
by (*metis diff-zero leD length-map length-upt nth-append*)
then show *?thesis*
using *left 4* **by** (*metis Cons-nth-drop-Suc Suc-diff-Suc diff-Suc-eq-diff-pred lt nth-Cons-Suc*)
next
case *5*
then show *?thesis*
using *lt contents-def* **by** *auto*
qed
qed
next
case *False*
moreover have $\lfloor map\ f\ [0..<t] \rfloor (Suc\ t := f\ t) = \lfloor map\ f\ [0..<Suc\ t] \rfloor$
proof
fix *x*
show $(\lfloor map\ f\ [0..<t] \rfloor (Suc\ t := f\ t))\ x = \lfloor map\ f\ [0..<Suc\ t] \rfloor\ x$
proof (*cases* $x < Suc\ t$)
case *True*
then show *?thesis*

using *contents-def*
by (*smt* (*verit*, *del-insts*) *diff-Suc-1 diff-zero fun-upd-apply length-map length-upt less-Suc-eq-0-disj less-Suc-eq-le less-imp-le-nat nat-neq-iff nth-map-upt*)
next
case *ge*: *False*
show *?thesis*
proof (*cases* $x = Suc\ t$)
case *True*
then show *?thesis*
using *contents-def*
by (*metis One-nat-def add-Suc diff-Suc-1 diff-zero fun-upd-same ge le-eq-less-or-eq length-map length-upt lessI less-Suc-eq-0-disj nth-map-upt plus-1-eq-Suc*)
next
case *False*
then have $x > Suc\ t$
using *ge* **by** *simp*
then show *?thesis*
using *contents-def* **by** *simp*
qed
qed
qed
ultimately show *?thesis*
by *simp*
qed

corollary *sem-cmd-plus'*:
assumes $j1 \neq j2$
and $j1 < k - 1$
and $j2 < k - 1$
and $j2 > 0$
and $length\ tps = k$
and *bit-symbols xs*
and *bit-symbols ys*
and $tps\ !\ j1 = (\lfloor xs \rfloor,\ Suc\ t)$
and $tps\ !\ j2 = (\lfloor map\ (sumdigit\ xs\ ys)\ [0..<t]\ @\ drop\ t\ ys \rfloor,\ Suc\ t)$
and $last\ tps = \lceil carry\ xs\ ys\ t \rceil$
and $tps' = tps$
$[j1 := (\lfloor xs \rfloor,\ Suc\ (Suc\ t)),$
$j2 := (\lfloor map\ (sumdigit\ xs\ ys)\ [0..<Suc\ t]\ @\ drop\ (Suc\ t)\ ys \rfloor,\ Suc\ (Suc\ t)),$
$length\ tps - 1 := \lceil carry\ xs\ ys\ (Suc\ t) \rceil]$
shows $sem\ (cmd\text{-}plus\ j1\ j2)\ (0,\ tps) = (if\ Suc\ t \le max\ (length\ xs)\ (length\ ys)\ then\ 0\ else\ 1,\ tps')$
proof −
have $tps\ !\ j1\ |{+}|\ 1 = (\lfloor xs \rfloor,\ Suc\ (Suc\ t))$
using *assms*(*8*) **by** *simp*
moreover have $tps\ !\ j2\ |{:=}|\ sumdigit\ xs\ ys\ t\ |{+}|\ 1 =$
$(\lfloor map\ (sumdigit\ xs\ ys)\ [0..<Suc\ t]\ @\ drop\ (Suc\ t)\ ys \rfloor,\ Suc\ (Suc\ t))$
using *contents-map-append-drop assms*(*9*) **by** *simp*
ultimately show *?thesis*
using *sem-cmd-plus*[*OF assms*(*1*−*10*)] *assms*(*11*) **by** *auto*
qed

The next Turing machine comprises just the command *cmd-plus*. It overwrites tape j_2 with the sum of the numbers on tape j_1 and j_2. The carry bit is maintained on the last tape.

definition *tm-plus* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
$tm\text{-}plus\ j1\ j2 \equiv [cmd\text{-}plus\ j1\ j2]$

lemma *tm-plus-tm*:
assumes $j2 > 0$ **and** $k \ge 2$ **and** $G \ge 4$
shows *turing-machine k G* (*tm-plus j1 j2*)
unfolding *tm-plus-def* **using** *assms*(*1*−*3*) *cmd-plus-def turing-machine-def* **by** *auto*

lemma *tm-plus-immobile*:
fixes *k* :: *nat*

```
  assumes j1 < k and j2 < k
  shows immobile (tm-plus j1 j2) k (Suc k)
proof –
  let ?M = tm-plus j1 j2
  { fix q :: nat and rs :: symbol list
    assume q: q < length ?M
    assume rs: length rs = Suc k
    then have len: length rs − 1 = k
      by simp
    have neq: k ≠ j1 k ≠ j2
      using assms by simp-all
    have ?M ! q = cmd-plus j1 j2
      using tm-plus-def q by simp
    moreover have (cmd-plus j1 j2) rs [!] k =
        (tosym ((todigit (rs ! j1) + todigit (rs ! j2) + todigit (last rs)) div 2), Stay)
      using cmd-plus-def rs len neq by fastforce
    ultimately have (cmd-plus j1 j2) rs [~] k = Stay
      by simp
  }
  then show ?thesis
    by (simp add: immobile-def tm-plus-def)
qed

lemma execute-tm-plus:
  assumes j1 ≠ j2
    and j1 < k − 1
    and j2 < k − 1
    and j2 > 0
    and length tps = k
    and bit-symbols xs
    and bit-symbols ys
    and t ≤ Suc (max (length xs) (length ys))
    and tps ! j1 = (⌊xs⌋, 1)
    and tps ! j2 = (⌊ys⌋, 1)
    and last tps = ⌈▷⌉
  shows execute (tm-plus j1 j2) (0, tps) t =
    (if t ≤ max (length xs) (length ys) then 0 else 1, tps
      [j1 := (⌊xs⌋, Suc t),
       j2 := (⌊map (sumdigit xs ys) [0..<t] @ drop t ys⌋, Suc t),
       length tps − 1 := ⌈carry xs ys t⌉])
  using assms(8)
proof (induction t)
  case 0
  have carry xs ys 0 = 1
    by simp
  moreover have map (sumdigit xs ys) [0..<0] @ drop 0 ys = ys
    by simp
  ultimately have tps = tps
      [j1 := (⌊xs⌋, Suc 0),
       j2 := (⌊map (sumdigit xs ys) [0..<0] @ drop 0 ys⌋, Suc 0),
       length tps − 1 := ⌈carry xs ys 0⌉]
    using assms
    by (metis One-nat-def add-diff-inverse-nat last-length less-nat-zero-code
      list-update-id nat-diff-split-asm plus-1-eq-Suc)
  then show ?case
    by simp
next
  case (Suc t)
  let ?M = tm-plus j1 j2
  have execute ?M (0, tps) (Suc t) = exe ?M (execute ?M (0, tps) t)
      (is - = exe ?M ?cfg)
    by simp
  also have ... = sem (cmd-plus j1 j2) ?cfg
```

```
    using Suc tm-plus-def exe-lt-length by simp
  also have ... = (if Suc t ≤ max (length xs) (length ys) then 0 else 1, tps
      [j1 := (⌊xs⌋, Suc (Suc t)),
       j2 := (⌊map (sumdigit xs ys) [0..<Suc t] @ drop (Suc t) ys⌋, Suc (Suc t)),
       length tps − 1 := ⌈carry xs ys (Suc t)⌉])
  proof −
    let ?tps = tps
      [j1 := (⌊xs⌋, Suc t),
       j2 := (⌊map (sumdigit xs ys) [0..<t] @ drop t ys⌋, Suc t),
       length tps − 1 := ⌈carry xs ys t⌉]
    let ?tps′ = ?tps
      [j1 := (⌊xs⌋, Suc (Suc t)),
       j2 := (⌊map (sumdigit xs ys) [0..<Suc t] @ drop (Suc t) ys⌋, Suc (Suc t)),
       length tps − 1 := ⌈carry xs ys (Suc t)⌉]
    have cfg: ?cfg = (0, ?tps)
      using Suc by simp
    have tps-k: length ?tps = k
      using assms(2,3,5) by simp
    have tps-j1: ?tps ! j1 = (⌊xs⌋, Suc t)
      using assms(1−3,5) by simp
    have tps-j2: ?tps ! j2 = (⌊map (sumdigit xs ys) [0..<t] @ drop t ys⌋, Suc t)
      using assms(1−3,5) by simp
    have tps-last: last ?tps = ⌈carry xs ys t⌉
      using assms
        by (metis One-nat-def carry.simps(1) diff-Suc-1 last-list-update length-list-update list-update-nonempty
prod.sel(2) tps-j1)
    then have sem (cmd-plus j1 j2) (0, ?tps) = (if Suc t ≤ max (length xs) (length ys) then 0 else 1, ?tps′)
      using sem-cmd-plus′[OF assms(1−4) tps-k assms(6,7) tps-j1 tps-j2 tps-last] assms(1−3)
      by (smt (verit, best) Suc.prems Suc-lessD assms(5) tps-k)
    then have sem (cmd-plus j1 j2) ?cfg = (if Suc t ≤ max (length xs) (length ys) then 0 else 1, ?tps′)
      using cfg by simp
    moreover have ?tps′ = tps
      [j1 := (⌊xs⌋, Suc (Suc t)),
       j2 := (⌊map (sumdigit xs ys) [0..<Suc t] @ drop (Suc t) ys⌋, Suc (Suc t)),
       length tps − 1 := ⌈carry xs ys (Suc t)⌉]
      using assms by (smt (verit) list-update-overwrite list-update-swap)
    ultimately show ?thesis
      by simp
  qed
  finally show ?case
    by simp
qed

lemma tm-plus-bounded-write:
  assumes j1 < k − 1
  shows bounded-write (tm-plus j1 j2) (k − 1) 4
  using assms cmd-plus-def tm-plus-def bounded-write-def by simp

lemma carry-max-length:
  assumes bit-symbols xs and bit-symbols ys
  shows carry xs ys (Suc (max (length xs) (length ys))) = 𝟬
proof −
  let ?t = max (length xs) (length ys)
  have carry xs ys (Suc ?t) = tosym ((todigit (digit xs ?t) + todigit (digit ys ?t) + todigit (carry xs ys ?t)) div
2)
    by simp
  then have carry xs ys (Suc ?t) = tosym (todigit (carry xs ys ?t) div 2)
    using digit-def by simp
  moreover have carry xs ys ?t ≤ 𝟭
    using carry-le assms by fastforce
  ultimately show ?thesis
    by simp
qed
```

corollary *execute-tm-plus-halt*:
  assumes $j1 \neq j2$
    and $j1 < k - 1$
    and $j2 < k - 1$
    and $j2 > 0$
    and *length tps* $= k$
    and *bit-symbols xs*
    and *bit-symbols ys*
    and $t = Suc\ (max\ (length\ xs)\ (length\ ys))$
    and $tps\ !\ j1 = (\lfloor xs \rfloor,\ 1)$
    and $tps\ !\ j2 = (\lfloor ys \rfloor,\ 1)$
    and *last tps* $= \lceil \triangleright \rceil$
  shows *execute* (*tm-plus j1 j2*) $(0,\ tps)\ t =$
    $(1,\ tps$
      $[j1 := (\lfloor xs \rfloor,\ Suc\ t),$
       $j2 := (\lfloor map\ (sumdigit\ xs\ ys)\ [0..<t] \rfloor,\ Suc\ t),$
       $length\ tps - 1 := \lceil \mathbf{0} \rceil])$
proof −
  have *execute* (*tm-plus j1 j2*) $(0,\ tps)\ t =$
    $(1,\ tps$
      $[j1 := (\lfloor xs \rfloor,\ Suc\ t),$
       $j2 := (\lfloor map\ (sumdigit\ xs\ ys)\ [0..<t]\ @\ drop\ t\ ys \rfloor,\ Suc\ t),$
       $length\ tps - 1 := \lceil carry\ xs\ ys\ t \rceil])$
    using *assms*(*8*) *execute-tm-plus*[*OF assms*(*1*−*7*) - *assms*(*9*−*11*)] *Suc-leI Suc-n-not-le-n lessI*
    by *presburger*
  then have *execute* (*tm-plus j1 j2*) $(0,\ tps)\ t =$
    $(1,\ tps$
      $[j1 := (\lfloor xs \rfloor,\ Suc\ t),$
       $j2 := (\lfloor map\ (sumdigit\ xs\ ys)\ [0..<t] \rfloor,\ Suc\ t),$
       $length\ tps - 1 := \lceil carry\ xs\ ys\ t \rceil])$
    using *assms*(*8*) **by** *simp*
  then show *execute* (*tm-plus j1 j2*) $(0,\ tps)\ t =$
    $(1,\ tps$
      $[j1 := (\lfloor xs \rfloor,\ Suc\ t),$
       $j2 := (\lfloor map\ (sumdigit\ xs\ ys)\ [0..<t] \rfloor,\ Suc\ t),$
       $length\ tps - 1 := \lceil \mathbf{0} \rceil])$
    using *assms*(*8*) *carry-max-length*[*OF assms*(*6*,*7*)] **by** *metis*
qed

lemma *transforms-tm-plusI*:
  assumes $j1 \neq j2$
    and $j1 < k - 1$
    and $j2 < k - 1$
    and $j2 > 0$
    and *length tps* $= k$
    and *bit-symbols xs*
    and *bit-symbols ys*
    and $t = Suc\ (max\ (length\ xs)\ (length\ ys))$
    and $tps\ !\ j1 = (\lfloor xs \rfloor,\ 1)$
    and $tps\ !\ j2 = (\lfloor ys \rfloor,\ 1)$
    and *last tps* $= \lceil \triangleright \rceil$
    and $tps' = tps$
      $[j1 := (\lfloor xs \rfloor,\ Suc\ t),$
       $j2 := (\lfloor map\ (sumdigit\ xs\ ys)\ [0..<t] \rfloor,\ Suc\ t),$
       $length\ tps - 1 := \lceil \mathbf{0} \rceil]$
  shows *transforms* (*tm-plus j1 j2*) *tps t tps'*
  using *assms execute-tm-plus-halt*[*OF assms*(*1*−*11*)] *tm-plus-def transforms-def transits-def*
  by *auto*

The next Turing machine removes the memorization tape from *tm-plus*.

definition *tm-plus'* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  $tm\text{-}plus'\ j1\ j2 \equiv cartesian\ (tm\text{-}plus\ j1\ j2)\ 4$

lemma *tm-plus′-tm*:
  assumes *j2* > *0* **and** *k* ≥ *2* **and** *G* ≥ *4*
  shows *turing-machine k G* (*tm-plus′ j1 j2*)
  unfolding *tm-plus′-def* **using** *assms cartesian-tm tm-plus-tm* **by** *simp*

lemma *transforms-tm-plus′I* [*transforms-intros*]:
  fixes *k t* :: *nat* **and** *j1 j2* :: *tapeidx* **and** *tps tps′* :: *tape list* **and** *xs zs* :: *symbol list*
  assumes *j1* ≠ *j2*
    and *j1* < *k*
    and *j2* < *k*
    and *j2* > *0*
    and *length tps* = *k*
    and *bit-symbols xs*
    and *bit-symbols ys*
    and *t* = *Suc* (*max* (*length xs*) (*length ys*))
    and *tps* ! *j1* = (⌊*xs*⌋, *1*)
    and *tps* ! *j2* = (⌊*ys*⌋, *1*)
    and *tps′* = *tps*
      [*j1* := (⌊*xs*⌋, *Suc t*),
       j2 := (⌊*map* (*sumdigit xs ys*) [*0*..<*t*]⌋, *Suc t*)]
  shows *transforms* (*tm-plus′ j1 j2*) *tps t tps′*
proof −
  let *?tps* = *tps* @ [⌈▷⌉]
  let *?tps′* = *?tps*
      [*j1* := (⌊*xs*⌋, *Suc t*),
       j2 := (⌊*map* (*sumdigit xs ys*) [*0*..<*t*]⌋, *Suc t*),
       length ?tps − *1* := ⌈**0**⌉]
  let *?M* = *tm-plus j1 j2*

  have *1*: *length ?tps* = *Suc k*
    using *assms*(*5*) **by** *simp*
  have *2*: *?tps* ! *j1* = (⌊*xs*⌋, *1*)
    by (*simp add*: *assms*(*9*) *assms*(*2*) *assms*(*5*) *nth-append*)
  have *3*: *?tps* ! *j2* = (⌊*ys*⌋, *1*)
    by (*simp add*: *assms*(*10*) *assms*(*3*) *assms*(*5*) *nth-append*)
  have *4*: *last ?tps* = ⌈▷⌉
    by *simp*
  have *5*: *k* ≥ *2*
    using *assms*(*3*,*4*) **by** *simp*
  have *transforms* (*tm-plus j1 j2*) *?tps t ?tps′*
    using *transforms-tm-plusI*[*OF assms*(*1*) - - *assms*(*4*) *1 assms*(*6*,*7*,*8*) *2 3 4*, *of ?tps′*] *assms*(*2*,*3*)
    by *simp*
  moreover have *?tps′* = *tps′* @ [⌈**0**⌉]
    using *assms* **by** (*simp add*: *list-update-append*)
  ultimately have *transforms* (*tm-plus j1 j2*) (*tps* @ [⌈▷⌉]) *t* (*tps′* @ [⌈**0**⌉])
    by *simp*
  moreover have *turing-machine* (*Suc k*) *4 ?M*
    using *tm-plus-tm assms* **by** *simp*
  moreover have *immobile ?M k* (*Suc k*)
    using *tm-plus-immobile assms* **by** *simp*
  moreover have *bounded-write* (*tm-plus j1 j2*) *k 4*
    using *tm-plus-bounded-write*[*of j1 Suc k*] *assms*(*2*) **by** *simp*
  ultimately have *transforms* (*cartesian* (*tm-plus j1 j2*) *4*) *tps t tps′*
    using *cartesian-transforms-onesie*[**where** *?M*=*?M* **and** *?b*=*4*] *assms*(*5*) *5*
    by *simp*
  then show *?thesis*
    using *tm-plus′-def* **by** *simp*
qed

The next Turing machine is the one we actually use to add two numbers. After computing the sum by running *tm-plus′*, it removes trailing zeros and performs a carriage return on the tapes j_1 and j_2.

definition *tm-add* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**

```
  tm-add j1 j2 ≡
    tm-plus' j1 j2 ;;
    tm-lconst-until j2 j2 {h. h ≠ 𝟬 ∧ h ≠ □} □ ;;
    tm-cr j1 ;;
    tm-cr j2

lemma tm-add-tm:
  assumes j2 > 0 and k ≥ 2 and G ≥ 4 and j2 < k
  shows turing-machine k G (tm-add j1 j2)
  unfolding tm-add-def using tm-plus'-tm tm-lconst-until-tm tm-cr-tm assms by simp

locale turing-machine-add =
  fixes j1 j2 :: tapeidx
begin

definition tm1 ≡ tm-plus' j1 j2
definition tm2 ≡ tm1 ;; tm-lconst-until j2 j2 {h. h ≠ 𝟬 ∧ h ≠ □} □
definition tm3 ≡ tm2 ;; tm-cr j1
definition tm4 ≡ tm3 ;; tm-cr j2

lemma tm4-eq-tm-add: tm4 = tm-add j1 j2
  using tm4-def tm3-def tm2-def tm1-def tm-add-def by simp

context
  fixes x y k :: nat and tps0 :: tape list
  assumes jk: j1 ≠ j2 j1 < k j2 < k j2 > 0 k = length tps0
  assumes tps0:
    tps0 ! j1 = (⌊canrepr x⌋, 1)
    tps0 ! j2 = (⌊canrepr y⌋, 1)
begin

abbreviation xs ≡ canrepr x

abbreviation ys ≡ canrepr y

lemma xs: bit-symbols xs
  using bit-symbols-canrepr by simp

lemma ys: bit-symbols ys
  using bit-symbols-canrepr by simp

abbreviation n ≡ Suc (max (length xs) (length ys))

abbreviation m ≡ length (canrepr (num xs + num ys))

definition tps1 ≡ tps0
  [j1 := (⌊xs⌋, Suc n),
   j2 := (⌊map (sumdigit xs ys) [0..<n]⌋, Suc n)]

lemma tm1 [transforms-intros]:
  assumes ttt = n
  shows transforms tm1 tps0 ttt tps1
  unfolding tm1-def
proof (tform tps: jk xs ys tps0 time: assms)
  show tps1 = tps0
    [j1 := (⌊xs⌋, Suc ttt),
     j2 := (⌊map (sumdigit xs ys) [0..<ttt]⌋, Suc ttt)]
    using tps1-def assms by simp
qed

definition tps2 ≡ tps0
  [j1 := (⌊xs⌋, Suc n),
   j2 := (⌊canrepr (num xs + num ys)⌋, m)]
```

lemma *contents-canlen*:
  assumes *bit-symbols zs*
  shows $\lfloor zs \rfloor$ (*canlen zs*) $\in \{h.\ h \neq \mathbf{0} \wedge \Box < h\}$
  using *assms contents-def canlen-le-length canlen-one* **by** *auto*

lemma *tm2* [*transforms-intros*]:
  assumes *ttt* = *n* + *Suc* (*Suc n* − *canlen* (*map* (*sumdigit xs ys*) [*0..<n*]))
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def*
proof (*tform tps*: *tps1-def jk xs ys tps0*)
  let *?zs* = *map* (*sumdigit xs ys*) [*0..<n*]
  have *bit-symbols ?zs*
    using *sumdigit-bit-symbols* **by** *blast*
  let *?ln* = *Suc n* − *canlen ?zs*
  have *lneigh* ($\lfloor$*?zs*$\rfloor$, *Suc n*) $\{h.\ h \neq \mathbf{0} \wedge \Box < h\}$ *?ln*
  proof (*rule lneighI*)
    have $\lfloor$*?zs*$\rfloor$ (*canlen ?zs*) $\in \{h.\ h \neq \mathbf{0} \wedge \Box < h\}$
      using *contents-canlen*[*OF* ‹*bit-symbols ?zs*›] **by** *simp*
    moreover have *Suc n* − *?ln* = *canlen ?zs*
      by (*metis One-nat-def diff-Suc-1 diff-Suc-Suc diff-diff-cancel le-imp-less-Suc*
        length-map length-upt less-imp-le-nat canlen-le-length)
    ultimately have $\lfloor$*?zs*$\rfloor$ (*Suc n* − *?ln*) $\in \{h.\ h \neq \mathbf{0} \wedge \Box < h\}$
      by *simp*
    then show *fst* ($\lfloor$*?zs*$\rfloor$, *Suc n*) (*snd* ($\lfloor$*?zs*$\rfloor$, *Suc n*) − *?ln*) $\in \{h.\ h \neq \mathbf{0} \wedge \Box < h\}$
      by *simp*

    have $\lfloor$*?zs*$\rfloor$ (*Suc n* − *n′*) $\in \{\Box, \mathbf{0}\}$ **if** *n′* < *?ln* **for** *n′*
    proof (*cases Suc n* − *n′* ≤ *n*)
      case *True*
      moreover have *1*: *Suc n* − *n′* > *0*
        using *that* **by** *simp*
      ultimately have $\lfloor$*?zs*$\rfloor$ (*Suc n* − *n′*) = *?zs* ! (*Suc n* − *n′* − *1*)
        using *contents-def* **by** *simp*
      moreover have *Suc n* − *n′* − *1* < *length ?zs*
        using *that True* **by** *simp*
      moreover have *Suc n* − *n′* − *1* ≥ *canlen ?zs*
        using *that* **by** *simp*
      ultimately show *?thesis*
        using *canlen-at-ge*[*of ?zs*] **by** *simp*
    next
      case *False*
      then show *?thesis*
        by *simp*
    qed
    then have $\lfloor$*?zs*$\rfloor$ (*Suc n* − *n′*) $\notin \{h.\ h \neq \mathbf{0} \wedge \Box < h\}$ **if** *n′* < *?ln* **for** *n′*
      using *that* **by** *fastforce*
    then show *fst* ($\lfloor$*?zs*$\rfloor$, *Suc n*) (*snd* ($\lfloor$*?zs*$\rfloor$, *Suc n*) − *n′*) $\notin \{h.\ h \neq \mathbf{0} \wedge \Box < h\}$
        if *n′* < *?ln* **for** *n′*
      using *that* **by** *simp*
  qed
  then show *lneigh* (*tps1* ! *j2*) $\{h.\ h \neq \mathbf{0} \wedge h \neq \Box\}$ *?ln*
    using *assms tps1-def jk* **by** *simp*
  show *Suc n* − *canlen* (*map* (*sumdigit xs ys*) [*0..<n*]) ≤ *tps1* :#: *j2*
    Suc n − *canlen* (*map* (*sumdigit xs ys*) [*0..<n*]) ≤ *tps1* :#: *j2*
    using *assms tps1-def jk* **by** *simp-all*

  have *num-zs*: *num ?zs* = *num xs* + *num ys*
    using *assms num-sumdigit-eq-sum″ xs ys* **by** *simp*
  then have *canrepr*: *canrepr* (*num xs* + *num ys*) = *take* (*canlen ?zs*) *?zs*
    using *canrepr-take-canlen* ‹*bit-symbols ?zs*› **by** *blast*
  have *len-canrepr*: *length* (*canrepr* (*num xs* + *num ys*)) = *canlen ?zs*
    using *num-zs length-canrepr-canlen sumdigit-bit-symbols* **by** *blast*

have *lconstplant* ($\lfloor$*?zs*$\rfloor$, *Suc n*) □ *?ln* =
  ($\lfloor$*canrepr* (*num xs* + *num ys*)$\rfloor$, *m*)
 (**is** *lconstplant ?tp* □ *?ln* = -)
proof −
 have (*if Suc n* − *?ln* < *i* ∧ *i* ≤ *Suc n then* □ *else* $\lfloor$*?zs*$\rfloor$ *i*) =
   $\lfloor$*take* (*canlen ?zs*) *?zs*$\rfloor$ *i*
   (**is** *?lhs* = *?rhs*)
  for *i*
 proof −
  consider
    i = *0*
   | *i* > *0* ∧ *i* ≤ *canlen ?zs*
   | *i* > *canlen ?zs* ∧ *i* ≤ *Suc n*
   | *i* > *canlen ?zs* ∧ *i* > *Suc n*
   by *linarith*
  then show *?thesis*
  proof (*cases*)
   case *1*
   then show *?thesis*
    by *simp*
  next
   case *2*
   then have *i* ≤ *Suc n* − *?ln*
    using *canlen-le-length*
    by (*metis diff-diff-cancel diff-zero le-imp-less-Suc length-map length-upt less-imp-le-nat*)
   then have *lhs*: *?lhs* = $\lfloor$*?zs*$\rfloor$ *i*
    by *simp*
   have *take* (*canlen ?zs*) *?zs* ! (*i* − *1*) = *?zs* ! (*i* − *1*)
    using *2* **by** (*metis Suc-diff-1 Suc-less-eq le-imp-less-Suc nth-take*)
   then have *?rhs* = $\lfloor$*?zs*$\rfloor$ *i*
    using *2 contents-inbounds len-canrepr local.canrepr not-le canlen-le-length*
    by (*metis add-diff-inverse-nat add-leE*)
   then show *?thesis*
    using *lhs* **by** *simp*
  next
   case *3*
   then have *Suc n* − *?ln* < *i* ∧ *i* ≤ *Suc n*
    by (*metis diff-diff-cancel less-imp-le-nat less-le-trans*)
   then have *?lhs* = *0*
    by *simp*
   moreover have *?rhs* = *0*
    using *3 contents-outofbounds len-canrepr canrepr* **by** *metis*
   ultimately show *?thesis*
    by *simp*
  next
   case *4*
   then have *?lhs* = *0*
    by *simp*
   moreover have *?rhs* = *0*
    using *4 contents-outofbounds len-canrepr canrepr* **by** *metis*
   ultimately show *?thesis*
    by *simp*
  qed
 qed
 then have (λ*i*. *if Suc n* − *?ln* < *i* ∧ *i* ≤ *Suc n then* □ *else* $\lfloor$*?zs*$\rfloor$ *i*) =
   $\lfloor$*canrepr* (*num xs* + *num ys*)$\rfloor$
  using *canrepr* **by** *simp*
 moreover have *fst ?tp* = $\lfloor$*?zs*$\rfloor$
  by *simp*
 ultimately have (λ*i*. *if Suc n* − *?ln* < *i* ∧ *i* ≤ *Suc n then 0 else fst ?tp i*) =
   $\lfloor$*canrepr* (*num xs* + *num ys*)$\rfloor$ **by** *metis*
 moreover have *Suc n* − *?ln* = *m*

using *len-canrepr*
by (*metis add-diff-inverse-nat diff-add-inverse2 diff-is-0-eq diff-zero le-imp-less-Suc length-map length-upt less-imp-le-nat less-numeral-extra*(*3*) *canlen-le-length zero-less-diff*)
ultimately show *?thesis*
using *lconstplant*[*of ?tp 0 ?ln*] **by** *simp*
qed
then show *tps2* = *tps1*
[*j2* := *tps1* ! *j2* |−| *?ln*,
j2 := *lconstplant* (*tps1* ! *j2*) *0* *?ln*]
using *tps2-def tps1-def jk* **by** *simp*

show *ttt* = *n* + *Suc ?ln*
using *assms* **by** *simp*
qed

definition *tps3* ≡ *tps0*
[*j1* := (⌊*xs*⌋, *1*),
j2 := (⌊*canrepr* (*num xs* + *num ys*)⌋, *m*)]

lemma *tm3* [*transforms-intros*]:
assumes *ttt* = *n* + *Suc* (*Suc n* − *canlen* (*map* (*sumdigit xs ys*) [*0*..<*n*])) + *Suc n* + *2*
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def*
proof (*tform tps*: *tps2-def jk xs ys tps0 time*: *assms tps2-def jk*)
show *clean-tape* (*tps2* ! *j1*)
using *tps2-def jk xs*
by (*metis clean-tape-ncontents nth-list-update-eq nth-list-update-neq*)
show *tps3* = *tps2*[*j1* := *tps2* ! *j1* |#=| *1*]
using *tps3-def tps2-def jk* **by** (*simp add*: *list-update-swap*)
qed

definition *tps4* ≡ *tps0*
[*j1* := (⌊*xs*⌋, *1*),
j2 := (⌊*canrepr* (*num xs* + *num ys*)⌋, *1*)]

lemma *tm4*:
assumes *ttt* = *n* + *Suc* (*Suc n* − *canlen* (*map* (*sumdigit xs ys*) [*0*..<*n*])) + *Suc n* + *2* + *m* + *2*
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform tps*: *tps3-def jk xs ys tps0 time*: *assms tps3-def jk*)
show *clean-tape* (*tps3* ! *j2*)
using *tps3-def tps2-def jk tps0*(*1*) **by** (*metis clean-tape-ncontents list-update-id nth-list-update-eq*)
show *tps4* = *tps3*[*j2* := *tps3* ! *j2* |#=| *1*]
using *tps4-def tps3-def jk* **by** *simp*
qed

lemma *tm4′*:
assumes *ttt* = *3* ∗ *max* (*length xs*) (*length ys*) + *10*
shows *transforms tm4 tps0 ttt tps4*
proof −
let *?zs* = *map* (*sumdigit xs ys*) [*0*..<*n*]
have *num ?zs* = *num xs* + *num ys*
using *num-sumdigit-eq-sum″ xs ys* **by** *simp*
then have *1*: *length* (*canrepr* (*num xs* + *num ys*)) = *canlen ?zs*
using *length-canrepr-canlen sumdigit-bit-symbols* **by** *blast*
moreover have *length ?zs* = *n*
by *simp*
ultimately have *m* ≤ *n*
by (*metis canlen-le-length*)

have *n* + *Suc* (*Suc n* − *canlen ?zs*) + *Suc n* + *2* + *m* + *2* =
n + *Suc* (*Suc n* − *m*) + *Suc n* + *2* + *m* + *2*
using *1* **by** *simp*

```
  also have ... = n + Suc (Suc n − m) + Suc n + 4 + m
    by simp
  also have ... = n + Suc (Suc n) − m + Suc n + 4 + m
    using ‹m ≤ n› by simp
  also have ... = n + Suc (Suc n) + Suc n + 4
    using ‹m ≤ n› by simp
  also have ... = 3 ∗ n + 7
    by simp
  also have ... = ttt
    using assms by simp
  finally have n + Suc (Suc n − canlen ?zs) + Suc n + 2 + m + 2 = ttt .
  then show ?thesis
    using tm4 by simp
qed

definition tps4' ≡ tps0
  [j2 := (⌊x + y⌋N, 1)]

lemma tm4'':
  assumes ttt = 3 ∗ max (nlength x) (nlength y) + 10
  shows transforms tm4 tps0 ttt tps4'
proof −
  have canrepr (num xs + num ys) = canrepr (x + y)
    by (simp add: canrepr)
  then show ?thesis
    using assms tps0(1) tps4'-def tps4-def tm4' by (metis list-update-id)
qed

end

end

lemma transforms-tm-addI [transforms-intros]:
  fixes j1 j2 :: tapeidx
  fixes x y k ttt :: nat and tps tps' :: tape list
  assumes j1 ≠ j2 j1 < k j2 < k j2 > 0 k = length tps
  assumes
    tps ! j1 = (⌊canrepr x⌋, 1)
    tps ! j2 = (⌊canrepr y⌋, 1)
  assumes ttt = 3 ∗ max (nlength x) (nlength y) + 10
  assumes tps' = tps
    [j2 := (⌊x + y⌋N, 1)]
  shows transforms (tm-add j1 j2) tps ttt tps'
proof −
  interpret loc: turing-machine-add j1 j2 .
  show ?thesis
    using loc.tm4-eq-tm-add loc.tps4'-def loc.tm4'' assms by simp
qed
```

2.7.5 Multiplication

In this section we construct a Turing machine that multiplies two numbers, each on its own tape, and writes the result to another tape. It employs the common algorithm for multiplication, which for binary numbers requires only doubling a number and adding two numbers. For the latter we already have a TM; for the former we are going to construct one.

The common algorithm

For two numbers given as symbol sequences *xs* and *ys*, the common algorithm maintains an intermediate result, initialized with 0, and scans *xs* starting from the most significant digit. In each step the intermediate result is multiplied by two, and if the current digit of *xs* is **1**, the value of *ys* is added to the intermediate result.

fun *prod* :: *symbol list* ⇒ *symbol list* ⇒ *nat* ⇒ *nat* **where**
  prod xs ys 0 = *0* |
  prod xs ys (*Suc i*) = *2* ∗ *prod xs ys i* + (*if xs* ! (*length xs* − *1* − *i*) = *3* *then num ys else 0*)

After i steps of the algorithm, the intermediate result is the product of *ys* and the i most significant bits of *xs*.

lemma *prod*:
  assumes $i \le$ *length xs*
  shows *prod xs ys i* = *num* (*drop* (*length xs* − *i*) *xs*) ∗ *num ys*
  using *assms*
proof (*induction i*)
  case *0*
  then show *?case*
    using *num-def* **by** *simp*
next
  case (*Suc i*)
  then have $i <$ *length xs*
    by *simp*
  then have *drop* (*length xs* − *Suc i*) *xs* = (*xs* ! (*length xs* − *1* − *i*)) # *drop* (*length xs* − *i*) *xs*
    by (*metis Cons-nth-drop-Suc Suc-diff-Suc diff-Suc-eq-diff-pred*
      diff-Suc-less gr-implies-not0 length-greater-0-conv list.size(*3*))
  then show *?case*
    using *num-Cons Suc* **by** *simp*
qed

After *length xs* steps, the intermediate result is the final result:

corollary *prod-eq-prod*: *prod xs ys* (*length xs*) = *num xs* ∗ *num ys*
  using *prod* **by** *simp*

definition *prod′* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
  prod′ x y i ≡ *prod* (*canrepr x*) (*canrepr y*) *i*

lemma *prod′*: *prod′ x y* (*nlength x*) = $x * y$
  using *prod-eq-prod prod′-def* **by** (*simp add*: *canrepr*)

Multiplying by two

Since we represent numbers with the least significant bit at the left, a multiplication by two is a right shift with a **0** inserted as the least significant digit. The next command implements the right shift. It scans the tape j and memorizes the current symbol on the last tape. It only writes the symbols **0** and **1**.

definition *cmd-double* :: *tapeidx* ⇒ *command* **where**
  cmd-double j rs ≡
    (*if rs* ! *j* = □ *then 1 else 0*,
     (*map* (λ*i*.
       if i = *j then*
         if last rs = ▷ ∧ *rs* ! *j* = □ *then* (*rs* ! *i*, *Right*)
         else (*tosym* (*todigit* (*last rs*)), *Right*)
       else if i = *length rs* − *1 then* (*tosym* (*todigit* (*rs* ! *j*)), *Stay*)
       else (*rs* ! *i*, *Stay*)) [*0*..<*length rs*]))

lemma *turing-command-double*:
  assumes $k \ge 2$ **and** $G \ge 4$ **and** $j > 0$ **and** $j < k - 1$
  shows *turing-command k 1 G* (*cmd-double j*)
proof
  show ⋀*gs*. *length gs* = *k* ⟹ *length* ([!!] *cmd-double j gs*) = *length gs*
    using *cmd-double-def* **by** *simp*
  show ⋀*gs*. *length gs* = *k* ⟹ *0* < *k* ⟹ *cmd-double j gs* [.] *0* = *gs* ! *0*
    using *assms cmd-double-def* **by** *simp*
  show *cmd-double j gs* [.] *j′* < *G*
    if *length gs* = *k* ⋀*i*. *i* < *length gs* ⟹ *gs* ! *i* < *G j′* < *length gs*
    for *j′ gs*
  proof −

    consider $j' = j \mid j' = k - 1 \mid j' \neq j \wedge j' \neq k - 1$
      by *auto*
    then show *?thesis*
    proof (*cases*)
      case *1*
      then have *cmd-double j gs* [!] $j' =$
          (*if last gs* $= \triangleright \wedge$ *gs* ! $j = \Box$ *then* (*gs* ! *j*, *Right*)
           else (*tosym* (*todigit* (*last gs*)), *Right*))
        using *cmd-double-def assms*(*1*,*4*) *that*(*1*) **by** *simp*
      then have *cmd-double j gs* [.] $j' =$
          (*if last gs* $= \triangleright \wedge$ *gs* ! $j = \Box$ *then gs* ! *j else tosym* (*todigit* (*last gs*)))
        by *simp*
      then show *?thesis*
        using *that assms* **by** *simp*
    next
      case *2*
      then have *cmd-double j gs* [!] $j' =$ (*tosym* (*todigit* (*gs* ! *j*)), *Stay*)
        using *cmd-double-def assms*(*1*,*4*) *that*(*1*) **by** *simp*
      then show *?thesis*
        using *assms* **by** *simp*
    next
      case *3*
      then show *?thesis*
        using *cmd-double-def assms that* **by** *simp*
    qed
  qed
  show $\bigwedge$*gs. length gs* $= k \Longrightarrow$ [$*$] (*cmd-double j gs*) ≤ 1
    using *assms cmd-double-def* **by** *simp*
qed

lemma *sem-cmd-double-0*:
  assumes $j < k$
    and *bit-symbols xs*
    and $i \leq$ *length xs*
    and $i > 0$
    and *length tps* $=$ *Suc k*
    and *tps* ! $j = (\lfloor xs \rfloor, i)$
    and *tps* ! $k = \lceil z \rceil$
    and $tps' = tps$ [$j :=$ *tps* ! j |:=| *tosym* (*todigit z*) |+| *1*, $k := \lceil xs$! $(i - 1)\rceil$]
  shows *sem* (*cmd-double j*) (*0*, *tps*) $=$ (*0*, tps')
proof (*rule semI*)
  show *proper-command* (*Suc k*) (*cmd-double j*)
    using *cmd-double-def* **by** *simp*
  show *length tps* $=$ *Suc k*
    using *assms*(*5*) **.**
  show *length* $tps' =$ *Suc k*
    using *assms*(*5*,*8*) **by** *simp*
  show *fst* (*cmd-double j* (*read tps*)) $= 0$
    using *assms contents-def cmd-double-def tapes-at-read*$'$[*of j tps*]
    by (*smt* (*verit*, *del-insts*) *One-nat-def Suc-le-lessD Suc-le-mono Suc-pred fst-conv*
      less-imp-le-nat snd-conv zero-neq-numeral)
  show *act* (*cmd-double j* (*read tps*) [!] j') (*tps* ! j') $= tps'$! j'
      if $j' <$ *Suc k* **for** j'
  proof $-$
    define *rs* **where** *rs* $=$ *read tps*
    then have *rsj*: *rs* ! $j = xs$! $(i - 1)$
      using *assms tapes-at-read*$'$ *contents-inbounds*
      by (*metis fst-conv le-imp-less-Suc less-imp-le-nat snd-conv*)
    then have *rs23*: *rs* ! $j = \mathbf{0} \vee$ *rs* ! $j = \mathbf{1}$
      using *assms* **by** *simp*
    have *lenrs*: *length rs* $=$ *Suc k*
      by (*simp add*: *rs-def assms*(*5*) *read-length*)
    consider $j' = j \mid j' = k \mid j' \neq j \wedge j' \neq k$

```
      by auto
    then show ?thesis
    proof (cases)
      case 1
      then have j' < length rs
        using lenrs that by simp
      then have cmd-double j rs [!] j' =
          (if last rs = ▷ ∧ rs ! j = □ then (rs ! j, Right)
           else (tosym (todigit (last rs)), Right))
        using cmd-double-def that 1 by simp
      then have cmd-double j rs [!] j' = (tosym (todigit (last rs)), Right)
        using rs23 lenrs assms by auto
      moreover have last rs = z
        using lenrs assms(5,7) rs-def onesie-read[of z] tapes-at-read'[of - tps]
        by (metis diff-Suc-1 last-conv-nth length-0-conv lessI old.nat.distinct(2))
      ultimately show ?thesis
        using act-Right' rs-def 1 assms(1,5,8) by simp
    next
      case 2
      then have j' = length rs - 1 j' ≠ j j' < length rs
        using lenrs that assms(1) by simp-all
      then have (cmd-double j rs) [!] j' = (tosym (todigit (rs ! j)), Stay)
        using cmd-double-def by simp
      then have (cmd-double j rs) [!] j' = (xs ! (i - 1), Stay)
        using rsj rs23 by auto
      then show ?thesis
        using act-onesie rs-def 2 assms that by simp
    next
      case 3
      then have j' ≠ length rs - 1 j' ≠ j j' < length rs
        using lenrs that by simp-all
      then have (cmd-double j rs) [!] j' = (rs ! j', Stay)
        using cmd-double-def by simp
      then show ?thesis
        using act-Stay rs-def assms that 3 by simp
    qed
  qed
qed

lemma sem-cmd-double-1:
  assumes j < k
    and bit-symbols xs
    and i > length xs
    and length tps = Suc k
    and tps ! j = (⌊xs⌋, i)
    and tps ! k = ⌈z⌉
    and tps' = tps
      [j := tps ! j |:=| (if z = ▷ then □ else tosym (todigit z)) |+| 1,
       k := ⌈𝟬⌉]
  shows sem (cmd-double j) (0, tps) = (1, tps')
proof (rule semI)
  show proper-command (Suc k) (cmd-double j)
    using cmd-double-def by simp
  show length tps = Suc k
    using assms(4) .
  show length tps' = Suc k
    using assms(4,7) by simp
  show fst (cmd-double j (read tps)) = 1
    using assms contents-def cmd-double-def tapes-at-read'[of j tps] by simp
  have j < length tps
    using assms by simp
  show act (cmd-double j (read tps) [!] j') (tps ! j') = tps' ! j'
      if j' < Suc k for j'
```

```
  proof -
    define rs where rs = read tps
    then have rsj: rs ! j = □
      using tapes-at-read'[OF ‹j < length tps›] assms(1,3,4,5) by simp
    have lenrs: length rs = Suc k
      by (simp add: rs-def assms(4) read-length)
    consider j' = j | j' = k | j' ≠ j ∧ j' ≠ k
      by auto
    then show ?thesis
    proof (cases)
      case 1
      then have j' < length rs
        using lenrs that by simp
      then have cmd-double j rs [!] j' =
          (if last rs = ▷ ∧ rs ! j = □ then (rs ! j, Right)
           else (tosym (todigit (last rs)), Right))
        using cmd-double-def that 1 by simp
      moreover have last rs = z
        using assms onesie-read rs-def tapes-at-read'
        by (metis diff-Suc-1 last-conv-nth length-0-conv lenrs lessI nat.simps(3))
      ultimately have cmd-double j rs [!] j' =
          (if z = ▷ then (□, Right) else (tosym (todigit z), Right))
        using rsj 1 by simp
      then show ?thesis
        using act-Right' rs-def 1 assms(1,4,7) by simp
    next
      case 2
      then have j' = length rs - 1 j' ≠ j j' < length rs
        using lenrs that assms(1) by simp-all
      then have (cmd-double j rs) [!] j' = (tosym (todigit (rs ! j)), Stay)
        using cmd-double-def by simp
      then have (cmd-double j rs) [!] j' = (2, Stay)
        using rsj by auto
      then show ?thesis
        using act-onesie rs-def 2 assms that by simp
    next
      case 3
      then have j' ≠ length rs - 1 j' ≠ j j' < length rs
        using lenrs that by simp-all
      then have (cmd-double j rs) [!] j' = (rs ! j', Stay)
        using cmd-double-def by simp
      then show ?thesis
        using act-Stay rs-def assms that 3 by simp
    qed
  qed
qed
```

The next Turing machine consists just of the command *cmd-double*.

```
definition tm-double :: tapeidx ⇒ machine where
  tm-double j ≡ [cmd-double j]

lemma tm-double-tm:
  assumes k ≥ 2 and G ≥ 4 and j > 0 and j < k - 1
  shows turing-machine k G (tm-double j)
  using assms tm-double-def turing-command-double by auto

lemma execute-tm-double-0:
  assumes j < k
    and bit-symbols xs
    and length xs > 0
    and length tps = Suc k
    and tps ! j = (⌊xs⌋, 1)
    and tps ! k = ⌈▷⌉
```

```
    and t ≥ 1
    and t ≤ length xs
  shows execute (tm-double j) (0, tps) t =
    (0, tps [j := (⌊0 # take (t − 1) xs @ drop t xs⌋, Suc t), k := ⌈xs ! (t − 1)⌉])
  using assms(7,8)
proof (induction t rule: nat-induct-at-least)
  case base
  have execute (tm-double j) (0, tps) 1 = exe (tm-double j) (execute (tm-double j) (0, tps) 0)
    by simp
  also have ... = sem (cmd-double j) (execute (tm-double j) (0, tps) 0)
    using tm-double-def exe-lt-length by simp
  also have ... = sem (cmd-double j) (0, tps)
    by simp
  also have ... = (0, tps [j := tps ! j |:=| tosym (todigit 1) |+| 1, k := ⌈xs ! (1 − 1)⌉])
    using assms(7,8) sem-cmd-double-0[OF assms(1−2) - - assms(4,5,6)] by simp
  also have ... = (0, tps [j := (⌊0 # take (1 − 1) xs @ drop 1 xs⌋, Suc 1), k := ⌈xs ! (1 − 1)⌉])
  proof −
    have tps ! j |:=| tosym (todigit 1) |+| 1 = (⌊xs⌋, 1) |:=| tosym (todigit 1) |+| 1
      using assms(5) by simp
    also have ... = (⌊xs⌋(1 := tosym (todigit 1)), Suc 1)
      by simp
    also have ... = (⌊xs⌋(1 := 0), Suc 1)
      by auto
    also have ... = (⌊0 # drop 1 xs⌋, Suc 1)
    proof −
      have ⌊0 # drop 1 xs⌋ = ⌊xs⌋(1 := 0)
      proof
        fix i :: nat
        consider i = 0 | i = 1 | i > 1 ∧ i ≤ length xs | i > length xs
          by linarith
        then show ⌊0 # drop 1 xs⌋ i = (⌊xs⌋(1 := 0)) i
        proof (cases)
          case 1
          then show ?thesis
            by simp
        next
          case 2
          then show ?thesis
            by simp
        next
          case 3
          then have ⌊0 # drop 1 xs⌋ i = (0 # drop 1 xs) ! (i − 1)
            using assms(3) by simp
          also have ... = (drop 1 xs) ! (i − 2)
            using 3 by (metis Suc-1 diff-Suc-eq-diff-pred nth-Cons-pos zero-less-diff)
          also have ... = xs ! (Suc (i − 2))
            using 3 assms(5) by simp
          also have ... = xs ! (i − 1)
            using 3 by (metis Suc-1 Suc-diff-Suc)
          also have ... = ⌊xs⌋ i
            using 3 by simp
          also have ... = (⌊xs⌋(1 := 0)) i
            using 3 by simp
          finally show ?thesis .
        next
          case 4
          then show ?thesis
            by simp
        qed
      qed
      then show ?thesis
        by simp
    qed
```

```
      also have ... = (⌊0 # take (1 − 1) xs @ drop 1 xs⌋, Suc 1)
        by simp
      finally show ?thesis
        by auto
    qed
    finally show ?case .
  next
    case (Suc t)
    let ?xs = 0 # take (t − 1) xs @ drop t xs
    let ?z = xs ! (t − 1)
    let ?tps = tps
       [j := (⌊?xs⌋, Suc t),
        k := ⌈?z⌉]
    have lenxs: length ?xs = length xs
      using Suc by simp
    have 0: ?xs ! t = xs ! t
    proof −
      have t > 0
        using Suc by simp
      then have length (0 # take (t − 1) xs) = t
        using Suc by simp
      moreover have length (drop t xs) > 0
        using Suc by simp
      moreover have drop t xs ! 0 = xs ! t
        using Suc by simp
      ultimately have ((0 # take (t − 1) xs) @ drop t xs) ! t = xs ! t
        by (metis diff-self-eq-0 less-not-refl3 nth-append)
      then show ?thesis
        by simp
    qed
    have 1: bit-symbols ?xs
    proof −
      have bit-symbols (take (t − 1) xs)
        using assms(2) by simp
      moreover have bit-symbols (drop t xs)
        using assms(2) by simp
      moreover have bit-symbols [0]
        by simp
      ultimately have bit-symbols ([0] @ take (t − 1) xs @ drop t xs)
        using bit-symbols-append by presburger
      then show ?thesis
        by simp
    qed
    have 2: Suc t ≤ length ?xs
      using Suc by simp
    have 3: Suc t > 0
      using Suc by simp
    have 4: length ?tps = Suc k
      using assms by simp
    have 5: ?tps ! j = (⌊?xs⌋, Suc t)
      by (simp add: Suc-lessD assms(1,4) nat-neq-iff)
    have 6: ?tps ! k = ⌈?z⌉
      by (simp add: assms(4))
    have execute (tm-double j) (0, tps) (Suc t) = exe (tm-double j) (execute (tm-double j) (0, tps) t)
      by simp
    also have ... = sem (cmd-double j) (execute (tm-double j) (0, tps) t)
      using tm-double-def exe-lt-length Suc by simp
    also have ... = sem (cmd-double j) (0, ?tps)
      using Suc by simp
    also have ... = (0, ?tps [j := ?tps ! j |:=| tosym (todigit ?z) |+| 1, k := ⌈?xs ! (Suc t − 1)⌉])
      using sem-cmd-double-0[OF assms(1) 1 2 3 4 5 6] by simp
    also have ... = (0, ?tps [j := ?tps ! j |:=| tosym (todigit ?z) |+| 1, k := ⌈xs ! (Suc t − 1)⌉])
      using 0 by simp
```

```
also have ... = (0, tps [j := ?tps ! j |:=| tosym (todigit ?z) |+| 1, k := ⌈xs ! (Suc t − 1)⌉])
  using assms by (smt (verit) list-update-overwrite list-update-swap)
also have ... = (0, tps [j := (⌊?xs⌋, Suc t) |:=| tosym (todigit ?z) |+| 1, k := ⌈xs ! (Suc t − 1)⌉])
  using 5 by simp
also have ... = (0, tps
    [j := (⌊?xs⌋(Suc t := tosym (todigit ?z)), Suc (Suc t)),
     k := ⌈xs ! (Suc t − 1)⌉])
  by simp
also have ... = (0, tps
    [j := (⌊2 # take (Suc t − 1) xs @ drop (Suc t) xs⌋, Suc (Suc t)),
     k := ⌈xs ! (Suc t − 1)⌉])
proof −
  have ⌊?xs⌋(Suc t := tosym (todigit ?z)) = ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋
  proof
    fix i :: nat
    consider i = 0 | i > 0 ∧ i < Suc t | i = Suc t | i > Suc t ∧ i ≤ length xs | i > length xs
      by linarith
    then show (⌊?xs⌋(Suc t := tosym (todigit ?z))) i = ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋ i
    proof (cases)
      case 1
      then show ?thesis
        by simp
    next
      case 2
      then have lhs: (⌊?xs⌋(Suc t := tosym (todigit ?z))) i = ?xs ! (i − 1)
        using lenxs Suc by simp
      have ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋ i =
          (0 # take (Suc t − 1) xs @ drop (Suc t) xs) ! (i − 1)
        using Suc 2 by auto
      then have ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋ i =
          ((0 # take (Suc t − 1) xs) @ drop (Suc t) xs) ! (i − 1)
        by simp
      moreover have length (0 # take (Suc t − 1) xs) = Suc t
        using Suc.prems by simp
      ultimately have ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋ i =
          (0 # take (Suc t − 1) xs) ! (i − 1)
        using 2 by (metis Suc-diff-1 Suc-lessD nth-append)
      also have ... = (0 # take t xs) ! (i − 1)
        by simp
      also have ... = (0 # take (t − 1) xs @ [xs ! (t − 1)]) ! (i − 1)
        using Suc by (metis Suc-diff-le Suc-le-lessD Suc-lessD diff-Suc-1 take-Suc-conv-app-nth)
      also have ... = ((0 # take (t − 1) xs) @ [xs ! (t − 1)]) ! (i − 1)
        by simp
      also have ... = (0 # take (t − 1) xs) ! (i − 1)
        using 2 Suc
        by (metis One-nat-def Suc-leD Suc-le-eq Suc-pred length-Cons length-take less-Suc-eq-le
          min-absorb2 nth-append)
      also have ... = ((0 # take (t − 1) xs) @ drop t xs) ! (i − 1)
        using 2 Suc
        by (metis Suc-diff-1 Suc-diff-le Suc-leD Suc-lessD diff-Suc-1 length-Cons length-take
          less-Suc-eq min-absorb2 nth-append)
      also have ... = ?xs ! (i − 1)
        by simp
      finally have ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋ i = ?xs ! (i − 1) .
      then show ?thesis
        using lhs by simp
    next
      case 3
      moreover have ?z = 0 ∨ ?z = 1
        using ‹bit-symbols ?xs› Suc assms(2) by (metis Suc-diff-le Suc-leD Suc-le-lessD diff-Suc-1)
      ultimately have lhs: (⌊?xs⌋(Suc t := tosym (todigit ?z))) i = ?z
        by auto
      have ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋ i =
```

```
        ⌊(0 # take t xs) @ drop (Suc t) xs⌋ (Suc t)
        using 3 by simp
      also have ... = ((0 # take t xs) @ drop (Suc t) xs) ! t
        using 3 Suc by simp
      also have ... = (0 # take t xs) ! t
        using Suc by (metis Suc-leD length-Cons length-take lessI min-absorb2 nth-append)
      also have ... = xs ! (t − 1)
        using Suc by simp
      finally have ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋ i = ?z .
      then show ?thesis
        using lhs by simp
    next
      case 4
      then have (⌊?xs⌋(Suc t := tosym (todigit ?z))) i = ⌊?xs⌋ i
        by simp
      also have ... = ?xs ! (i − 1)
        using 4 by auto
      also have ... = ((0 # take (t − 1) xs) @ drop t xs) ! (i − 1)
        by simp
      also have ... = drop t xs ! (i − 1 − t)
        using 4 Suc
        by (smt (verit, ccfv-threshold) Cons-eq-appendI Suc-diff-1 Suc-leD
          add-diff-cancel-right′ bot-nat-0.extremum-uniqueI diff-diff-cancel
          length-append length-drop lenxs not-le not-less-eq nth-append)
      also have ... = xs ! (i − 1)
        using 4 Suc by simp
      finally have lhs: (⌊?xs⌋(Suc t := tosym (todigit ?z))) i = xs ! (i − 1) .
      have ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋ i =
          (0 # take (Suc t − 1) xs @ drop (Suc t) xs) ! (i − 1)
        using 4 by auto
      also have ... = ((0 # take t xs) @ drop (Suc t) xs) ! (i − 1)
        by simp
      also have ... = (drop (Suc t) xs) ! (i − 1 − Suc t)
        using Suc 4
        by (smt (verit) Suc-diff-1 Suc-leD Suc-leI bot-nat-0.extremum-uniqueI length-Cons length-take
          min-absorb2 not-le nth-append)
      also have ... = xs ! (i − 1)
        using Suc 4 Suc-lessE by fastforce
      finally have ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋ i = xs ! (i − 1) .
      then show ?thesis
        using lhs by simp
    next
      case 5
      then have (⌊?xs⌋(Suc t := tosym (todigit ?z))) i = ⌊?xs⌋ i
        using Suc by simp
      then have lhs: (⌊?xs⌋(Suc t := tosym (todigit ?z))) i = □
        using 5 contents-outofbounds lenxs by simp
      have length (0 # take (Suc t − 1) xs @ drop (Suc t) xs) = length xs
        using Suc by simp
      then have ⌊0 # take (Suc t − 1) xs @ drop (Suc t) xs⌋ i = □
        using 5 contents-outofbounds by simp
      then show ?thesis
        using lhs by simp
    qed
  qed
  then show ?thesis
    by simp
 qed
 finally show ?case .
qed

lemma execute-tm-double-1:
 assumes j < k
```

```
    and bit-symbols xs
    and length xs > 0
    and length tps = Suc k
    and tps ! j = (⌊xs⌋, 1)
    and tps ! k = ⌈▷⌉
  shows execute (tm-double j) (0, tps) (Suc (length xs)) =
    (1, tps [j := (⌊0 # xs⌋, length xs + 2), k := ⌈0⌉])
proof −
  let ?z = xs ! (length xs − 1)
  let ?xs = 0 # take (length xs − 1) xs
  have ?z ≠ ▷
    using assms(2,3) by (metis One-nat-def Suc-1 diff-less less-Suc-eq not-less-eq numeral-3-eq-3)
  have z23: ?z = 0 ∨ ?z = 1
    using assms(2,3) by (meson diff-less zero-less-one)
  have lenxs: length ?xs = length xs
    using assms(3) by (metis Suc-diff-1 diff-le-self length-Cons length-take min-absorb2)
  have 0: bit-symbols ?xs
    using assms(2) bit-symbols-append[of [0] take (length xs − 1) xs] by simp

  have execute (tm-double j) (0, tps) (length xs) =
    (0, tps
      [j := (⌊0 # take (length xs − 1) xs @ drop (length xs) xs⌋, Suc (length xs)),
       k := ⌈?z⌉])
    using execute-tm-double-0[OF assms(1−6), where ?t=length xs] assms(3) by simp
  then have ∗: execute (tm-double j) (0, tps) (length xs) =
    (0, tps [j := (⌊?xs⌋, Suc (length ?xs)), k := ⌈?z⌉])
    (is - = (0, ?tps))
    using lenxs by simp

  let ?i = Suc (length ?xs)
  have 1: ?i > length ?xs
    by simp
  have 2: length ?tps = Suc k
    using assms(4) by simp
  have 3: ?tps ! j = (⌊?xs⌋, ?i)
    using assms(1,4) by simp
  have 4: ?tps ! k = ⌈?z⌉
    using assms(4) by simp

  have execute (tm-double j) (0, tps) (Suc (length xs)) = exe (tm-double j) (0, ?tps)
    using ∗ by simp
  also have ... = sem (cmd-double j) (0, ?tps)
    using tm-double-def exe-lt-length by simp
  also have ... = (1, ?tps
      [j := ?tps ! j |:=| (if ?z = ▷ then □ else tosym (todigit ?z)) |+| 1,
       k := ⌈0⌉])
    using sem-cmd-double-1[OF assms(1) 0 1 2 3 4] by simp
  also have ... = (1, ?tps
      [j := ?tps ! j |:=| (tosym (todigit ?z)) |+| 1,
       k := ⌈0⌉])
    using ‹?z ≠ 1› by simp
  also have ... = (1, ?tps
      [j := (⌊?xs⌋, Suc (length ?xs)) |:=| (tosym (todigit ?z)) |+| 1,
       k := ⌈0⌉])
    using 3 by simp
  also have ... = (1, ?tps
      [j := (⌊?xs⌋, Suc (length ?xs)) |:=| ?z |+| 1,
       k := ⌈0⌉])
    using z23 One-nat-def Suc-1 add-2-eq-Suc' numeral-3-eq-3 by presburger
  also have ... = (1, tps
      [j := (⌊?xs⌋, Suc (length ?xs)) |:=| ?z |+| 1,
       k := ⌈0⌉])
    by (smt (verit) list-update-overwrite list-update-swap)
```

also have ... = (*1*, *tps*
[*j* := (⌊*?xs*⌋(*Suc* (*length ?xs*) := *?z*), *length ?xs* + *2*),
k := ⌈**0**⌉])
by *simp*
also have ... = (*1*, *tps*
[*j* := (⌊*?xs*⌋(*Suc* (*length ?xs*) := *?z*), *length xs* + *2*),
k := ⌈**0**⌉])
using *lenxs* **by** *simp*
also have ... = (*1*, *tps* [*j* := (⌊**0** # *xs*⌋, *length xs* + *2*), *k* := ⌈**0**⌉])
proof −
have ⌊*?xs*⌋(*Suc* (*length ?xs*) := *?z*) = ⌊**0** # *xs*⌋
proof
fix *i*
consider *i* = *0* | *i* > *0* ∧ *i* ≤ *length xs* | *i* = *Suc* (*length xs*) | *i* > *Suc* (*length xs*)
by *linarith*
then show (⌊*?xs*⌋(*Suc* (*length ?xs*) := *?z*)) *i* = ⌊**0** # *xs*⌋ *i*
proof (*cases*)
case *1*
then show *?thesis*
by *simp*
next
case *2*
then have (⌊*?xs*⌋(*Suc* (*length ?xs*) := *?z*)) *i* = ⌊*?xs*⌋ *i*
using *lenxs* **by** *simp*
also have ... = *?xs* ! (*i* − *1*)
using *2* **by** *auto*
also have ... = (**0** # *xs*) ! (*i* − *1*)
using *lenxs 2 assms*(*3*) **by** (*metis Suc-diff-1 Suc-le-lessD nth-take take-Suc-Cons*)
also have ... = ⌊**0** # *xs*⌋ *i*
using *2* **by** *simp*
finally show *?thesis* **.**
next
case *3*
then have *lhs*: (⌊*?xs*⌋(*Suc* (*length ?xs*) := *?z*)) *i* = *?z*
using *lenxs* **by** *simp*
have ⌊**0** # *xs*⌋ *i* = (**0** # *xs*) ! (*i* − *1*)
using *3 lenxs* **by** *simp*
also have ... = *xs* ! (*i* − *2*)
using *3 assms*(*3*) **by** *simp*
also have ... = *?z*
using *3* **by** *simp*
finally have ⌊**0** # *xs*⌋ *i* = *?z* **.**
then show *?thesis*
using *lhs* **by** *simp*
next
case *4*
then show *?thesis*
using *4 lenxs* **by** *simp*
qed
qed
then show *?thesis*
by *simp*
qed
finally show *?thesis* **.**
qed

lemma *execute-tm-double-Nil*:
assumes *j* < *k*
and *length tps* = *Suc k*
and *tps* ! *j* = (⌊[]⌋, *1*)
and *tps* ! *k* = ⌈▷⌉
shows *execute* (*tm-double j*) (*0*, *tps*) (*Suc 0*) =
(*1*, *tps* [*j* := (⌊[]⌋, *2*), *k* := ⌈**0**⌉])

```
proof −
  have execute (tm-double j) (0, tps) (Suc 0) = exe (tm-double j) (execute (tm-double j) (0, tps) 0)
    by simp
  also have ... = exe (tm-double j) (0, tps)
    by simp
  also have ... = sem (cmd-double j) (0, tps)
    using tm-double-def exe-lt-length by simp
  also have ... = (1, tps
      [j := tps ! j |:=| (if (1::nat) = 1 then 0 else tosym (todigit 1)) |+| 1,
       k := ⌈0⌉])
    using sem-cmd-double-1[OF assms(1) - - assms(2−4)] by simp
  also have ... = (1, tps [j := tps ! j |:=| □ |+| 1, k := ⌈0⌉])
    by simp
  also have ... = (1, tps [j := (⌊[]⌋, 1) |:=| □ |+| 1, k := ⌈0⌉])
    using assms(3) by simp
  also have ... = (1, tps [j := (⌊[]⌋(1 := □), 2), k := ⌈0⌉])
    by (metis fst-eqD one-add-one snd-eqD)
  also have ... = (1, tps [j := (⌊[]⌋, 2), k := ⌈0⌉])
    by (metis contents-outofbounds fun-upd-idem-iff list.size(3) zero-less-one)
  finally show ?thesis .
qed

lemma execute-tm-double:
  assumes j < k
    and length tps = Suc k
    and tps ! j = (⌊canrepr n⌋, 1)
    and tps ! k = ⌈▷⌉
  shows execute (tm-double j) (0, tps) (Suc (length (canrepr n))) =
    (1, tps [j := (⌊canrepr (2 ∗ n)⌋, length (canrepr n) + 2), k := ⌈0⌉])
proof (cases n = 0)
  case True
  then have canrepr n = []
    using canrepr-0 by simp
  then show ?thesis
    using execute-tm-double-Nil[OF assms(1−2) - assms(4)] assms(3) True
    by (metis add-2-eq-Suc' list.size(3) mult-0-right numeral-2-eq-2)
next
  case False
  let ?xs = canrepr n
  have num (0 # ?xs) = 2 ∗ num ?xs
    using num-Cons by simp
  then have num (0 # ?xs) = 2 ∗ n
    using canrepr by simp
  moreover have canonical (0 # ?xs)
  proof −
    have ?xs ≠ []
      using False canrepr canrepr-0 by metis
    then show ?thesis
      using canonical-Cons canonical-canrepr by simp
  qed
  ultimately have canrepr (2 ∗ n) = 0 # ?xs
    using canreprI by blast
  then show ?thesis
   using execute-tm-double-1[OF assms(1) - - assms(2) - assms(4)] assms(3) False canrepr canrepr-0 bit-symbols-canrepr
    by (metis length-greater-0-conv)
qed

lemma execute-tm-double-app:
  assumes j < k
    and length tps = k
    and tps ! j = (⌊canrepr n⌋, 1)
  shows execute (tm-double j) (0, tps @ [⌈▷⌉]) (Suc (length (canrepr n))) =
    (1, tps [j := (⌊canrepr (2 ∗ n)⌋, length (canrepr n) + 2)] @ [⌈0⌉])
```

```
proof -
  let ?tps = tps @ [⌈▷⌉]
  have length ?tps = Suc k
    using assms(2) by simp
  moreover have ?tps ! j = (⌊canrepr n⌋, 1)
    using assms(1,2,3) by (simp add: nth-append)
  moreover have ?tps ! k = ⌈▷⌉
    using assms(2) by (simp add: nth-append)
  moreover have tps [j := (⌊canrepr (2 ∗ n)⌋, length (canrepr n) + 2)] @ [⌈0⌉] =
      ?tps [j := (⌊canrepr (2 ∗ n)⌋, length (canrepr n) + 2), k := ⌈0⌉]
    using assms by (metis length-list-update list-update-append1 list-update-length)
  ultimately show ?thesis
    using assms execute-tm-double[OF assms(1), where ?tps=tps @ [⌈▷⌉]]
    by simp
qed

lemma transforms-tm-double:
  assumes j < k
    and length tps = k
    and tps ! j = (⌊canrepr n⌋, 1)
  shows transforms (tm-double j)
    (tps @ [⌈▷⌉])
    (Suc (length (canrepr n)))
    (tps [j := (⌊canrepr (2 ∗ n)⌋, length (canrepr n) + 2)] @ [⌈0⌉])
  using assms transforms-def transits-def tm-double-def execute-tm-double-app by auto

lemma tm-double-immobile:
  fixes k :: nat
  assumes j > 0 and j < k
  shows immobile (tm-double j) k (Suc k)
proof -
  let ?M = tm-double j
  { fix q :: nat and rs :: symbol list
    assume q: q < length ?M
    assume rs: length rs = Suc k
    then have len: length rs - 1 = k
      by simp
    have neq: k ≠ j
      using assms(2) by simp
    have ?M ! q = cmd-double j
      using tm-double-def q by simp
    moreover have (cmd-double j) rs [!] k = (tosym (todigit (rs ! j)), Stay)
      using cmd-double-def rs len neq by fastforce
    ultimately have (cmd-double j) rs [~] k = Stay
      by simp
  }
  then show ?thesis
    by (simp add: immobile-def tm-double-def)
qed

lemma tm-double-bounded-write:
  assumes j < k - 1
  shows bounded-write (tm-double j) (k - 1) 4
  using assms cmd-double-def tm-double-def bounded-write-def by simp
```

The next Turing machine removes the memorization tape.

```
definition tm-double' :: nat ⇒ machine where
  tm-double' j ≡ cartesian (tm-double j) 4

lemma tm-double'-tm:
  assumes j > 0 and k ≥ 2 and G ≥ 4 and j < k
  shows turing-machine k G (tm-double' j)
  unfolding tm-double'-def using assms cartesian-tm tm-double-tm by simp
```

lemma *transforms-tm-double′I* [*transforms-intros*]:
  assumes $j > 0$ **and** $j < k$
    and *length tps* = k
    and *tps* ! j = ($\lfloor$*canrepr n*$\rfloor$, *1*)
    and t = (*Suc* (*length* (*canrepr n*)))
    and *tps′* = *tps* [j := ($\lfloor$*canrepr* (*2* ∗ *n*)$\rfloor$, *length* (*canrepr n*) + *2*)]
  shows *transforms* (*tm-double′ j*) *tps t tps′*
  unfolding *tm-double′-def*
proof (*rule cartesian-transforms-onesie*)
  show *turing-machine* (*Suc k*) *4* (*tm-double j*)
    using *assms*(*1*,*2*) *tm-double-tm* **by** *simp*
  show *length tps* = *k 2* ≤ *k* (*1*::*nat*) < *4*
    using *assms* **by** *simp-all*
  show *bounded-write* (*tm-double j*) *k 4*
    by (*metis assms*(*2*) *diff-Suc-1 tm-double-bounded-write*)
  show *immobile* (*tm-double j*) *k* (*Suc k*)
    by (*simp add*: *assms*(*1*,*2*) *tm-double-immobile*)
  show *transforms* (*tm-double j*) (*tps* @ [⌈▷⌉]) *t* (*tps′* @ [⌈**0**⌉])
    using *assms transforms-tm-double* **by** *simp*
qed

The next Turing machine is the one we actually use to double a number. It runs *tm-double′* and performs a carriage return.

definition *tm-times2* :: *tapeidx* ⇒ *machine* **where**
  tm-times2 j ≡ *tm-double′ j* ;; *tm-cr j*

lemma *tm-times2-tm*:
  assumes $k \geq 2$ **and** $j > 0$ **and** $j < k$ **and** $G \geq 4$
  shows *turing-machine k G* (*tm-times2 j*)
  using *assms* **by** (*simp add*: *assms*(*1*) *tm-cr-tm tm-double′-tm tm-times2-def*)

lemma *transforms-tm-times2I* [*transforms-intros*]:
  assumes $j > 0$ **and** $j < k$
    and *length tps* = k
    and *tps* ! j = ($\lfloor n \rfloor_N$, *1*)
    and t = *5* + *2* ∗ *nlength n*
    and *tps′* = *tps* [j := ($\lfloor 2 * n \rfloor_N$, *1*)]
  shows *transforms* (*tm-times2 j*) *tps t tps′*
  unfolding *tm-times2-def*
proof (*tform tps*: *assms*)
  show *clean-tape* (*tps*[j := ($\lfloor 2 * n \rfloor_N$, *nlength n* + *2*)] ! *j*)
    using *clean-tape-ncontents assms* **by** *simp*
  show t = *Suc* (*nlength n*) + (*tps*[j := ($\lfloor 2 * n \rfloor_N$, *nlength n* + *2*)] :#: *j* + *2*)
    using *assms* **by** *simp*
qed

Multiplying arbitrary numbers

Before we can multiply arbitrary numbers we need just a few more lemmas.

lemma *num-drop-le-nu*: *num* (*drop j xs*) ≤ *num xs*
proof (*cases j* ≤ *length xs*)
  case *True*
  let *?ys* = *drop j xs*
  have *map-shift-upt*: *map* (λ*i*. *f* (*j* + *i*)) [*0*..<*l*] = *map f* [*j*..<*j* + *l*]
      for *f* :: *nat* ⇒ *nat* **and** *j l*
    by (*rule nth-equalityI*) *simp-all*

  have *num ?ys* = (∑ *i*←[*0*..<*length ?ys*]. *todigit* (*?ys* ! *i*) ∗ *2* ^ *i*)
    using *num-def* **by** *simp*
  also have ... = (∑ *i*←[*0*..<*length ?ys*]. *todigit* (*xs* ! (*j* + *i*)) ∗ *2* ^ *i*)
    by (*simp add*: *True*)

```
  also have ... ≤ 2 ^ j * (∑i←[0..<length ?ys]. todigit (xs ! (j + i)) * 2 ^ i)
    by simp
  also have ... = (∑i←[0..<length ?ys]. 2 ^ j * todigit (xs ! (j + i)) * 2 ^ i)
    by (simp add: mult.assoc sum-list-const-mult)
  also have ... = (∑i←[0..<length ?ys]. todigit (xs ! (j + i)) * 2 ^ (j + i))
    by (simp add: ab-semigroup-mult-class.mult-ac(1) mult.commute power-add)
  also have ... = (∑i←[j..<j + length ?ys]. todigit (xs ! i) * 2 ^ i)
    using map-shift-upt[of λi. todigit (xs ! i) * 2 ^ i j length ?ys] by simp
  also have ... ≤ (∑i←[0..<j]. todigit (xs ! i) * 2 ^ i) +
      (∑i←[j..<j + length ?ys]. todigit (xs ! i) * 2 ^ i)
    by simp
  also have ... = (∑i←[0..<j + length ?ys]. todigit (xs ! i) * 2 ^ i)
    by (metis (no-types, lifting) le-add2 le-add-same-cancel2 map-append sum-list.append upt-add-eq-append)
  also have ... = (∑i←[0..<length xs]. todigit (xs ! i) * 2 ^ i)
    by (simp add: True)
  also have ... = num xs
    using num-def by simp
  finally show ?thesis .
next
  case False
  then show ?thesis
    using canrepr canrepr-0 by (metis drop-all nat-le-linear zero-le)
qed

lemma nlength-prod-le-prod:
  assumes i ≤ length xs
  shows nlength (prod xs ys i) ≤ nlength (num xs * num ys)
  using prod[OF assms] num-drop-le-nu mult-le-mono1 nlength-mono by simp

corollary nlength-prod′-le-prod:
  assumes i ≤ nlength x
  shows nlength (prod′ x y i) ≤ nlength (x * y)
  using assms prod′-def nlength-prod-le-prod by (metis prod′ prod-eq-prod)

lemma two-times-prod:
  assumes i < length xs
  shows 2 * prod xs ys i ≤ num xs * num ys
proof −
  have 2 * prod xs ys i ≤ prod xs ys (Suc i)
    by simp
  also have ... = num (drop (length xs − Suc i) xs) * num ys
    using prod[of Suc i xs] assms by simp
  also have ... ≤ num xs * num ys
    using num-drop-le-nu by simp
  finally show ?thesis .
qed

corollary two-times-prod′:
  assumes i < nlength x
  shows 2 * prod′ x y i ≤ x * y
  using assms two-times-prod prod′-def by (metis prod′ prod-eq-prod)
```

The next Turing machine multiplies the numbers on tapes j_1 and j_2 and writes the result to tape j_3. It iterates over the binary digits on j_1 starting from the most significant digit. In each iteration it doubles the intermediate result on j_3. If the current digit is **1**, the number on j_2 is added to j_3.

```
definition tm-mult :: tapeidx ⇒ tapeidx ⇒ tapeidx ⇒ machine where
  tm-mult j1 j2 j3 ≡
    tm-right-until j1 {□} ;;
    tm-left j1 ;;
    WHILE [] ; λrs. rs ! j1 ≠ ▷ DO
      tm-times2 j3 ;;
      IF λrs. rs ! j1 = 𝟏 THEN
        tm-add j2 j3
```

```
      ELSE
        []
      ENDIF ;;
      tm-left j1
    DONE ;;
    tm-right j1

lemma tm-mult-tm:
  assumes j1 ≠ j2 j2 ≠ j3 j3 ≠ j1 and j3 > 0
  assumes k ≥ 2
    and G ≥ 4
    and j1 < k j2 < k j3 < k
  shows turing-machine k G (tm-mult j1 j2 j3)
  unfolding tm-mult-def
  using assms tm-left-tm tm-right-tm Nil-tm tm-add-tm tm-times2-tm tm-right-until-tm
    turing-machine-branch-turing-machine turing-machine-loop-turing-machine
  by simp

locale turing-machine-mult =
  fixes j1 j2 j3 :: tapeidx
begin

definition tm1 ≡ tm-right-until j1 {□}
definition tm2 ≡ tm1 ;; tm-left j1
definition tmIf ≡ IF λrs. rs ! j1 = 𝟭 THEN tm-add j2 j3 ELSE [] ENDIF
definition tmBody1 ≡ tm-times2 j3 ;; tmIf
definition tmBody ≡ tmBody1 ;; tm-left j1
definition tmWhile ≡ WHILE [] ; λrs. rs ! j1 ≠ ▷ DO tmBody DONE
definition tm3 ≡ tm2 ;; tmWhile
definition tm4 ≡ tm3 ;; tm-right j1

lemma tm4-eq-tm-mult: tm4 = tm-mult j1 j2 j3
  using tm1-def tm2-def tm3-def tm4-def tm-mult-def tmIf-def tmBody-def tmBody1-def tmWhile-def
  by simp

context
  fixes x y k :: nat and tps0 :: tape list
  assumes jk: j1 ≠ j2 j2 ≠ j3 j3 ≠ j1 j3 > 0 j1 < k j2 < k j3 < k length tps0 = k
  assumes tps0:
    tps0 ! j1 = (⌊x⌋_N, 1)
    tps0 ! j2 = (⌊y⌋_N, 1)
    tps0 ! j3 = (⌊0⌋_N, 1)
begin

definition tps1 ≡ tps0 [j1 := (⌊x⌋_N, Suc (nlength x))]

lemma tm1 [transforms-intros]:
  assumes t = Suc (nlength x)
  shows transforms tm1 tps0 t tps1
  unfolding tm1-def
proof (tform tps: assms tps0 tps1-def jk)
  show rneigh (tps0 ! j1) {□} (nlength x)
  proof (rule rneighI)
    show (tps0 ::: j1) (tps0 :#: j1 + nlength x) ∈ {□}
      by (simp add: tps0)
    show ⋀n'. n' < nlength x ⟹ (tps0 ::: j1) (tps0 :#: j1 + n') ∉ {□}
      using tps0 bit-symbols-canrepr contents-def by fastforce
  qed
qed

definition tps2 ≡ tps0 [j1 := (⌊x⌋_N, nlength x)]

lemma tm2 [transforms-intros]:
```

```
  assumes t = Suc (Suc (nlength x)) and tps' = tps2
  shows transforms tm2 tps0 t tps'
  unfolding tm2-def by (tform tps: assms tps1-def tps2-def jk)

definition tpsL t ≡ tps0
   [j1 := (⌊x⌋N, nlength x − t),
    j3 := (⌊prod' x y t⌋N, 1)]

definition tpsL1 t ≡ tps0
   [j1 := (⌊x⌋N, nlength x − t),
    j3 := (⌊2 ∗ prod' x y t⌋N, 1)]

definition tpsL2 t ≡ tps0
   [j1 := (⌊x⌋N, nlength x − t),
    j3 := (⌊prod' x y (Suc t)⌋N, 1)]

definition tpsL3 t ≡ tps0
   [j1 := (⌊x⌋N, nlength x − t − 1),
    j3 := (⌊prod' x y (Suc t)⌋N, 1)]

lemma tmIf [transforms-intros]:
  assumes t < nlength x and ttt = 12 + 3 ∗ nlength (x ∗ y)
  shows transforms tmIf (tpsL1 t) ttt (tpsL2 t)
  unfolding tmIf-def
proof (tform tps: assms tpsL1-def tps0 jk)
  have nlength y ≤ nlength (x ∗ y) ∧ nlength (2 ∗ prod' x y t) ≤ nlength (x ∗ y)
  proof
    have x > 0
      using assms(1) gr-implies-not-zero nlength-0 by auto
    then have y ≤ x ∗ y
      by simp
    then show nlength y ≤ nlength (x ∗ y)
      using nlength-mono by simp
    show nlength (2 ∗ prod' x y t) ≤ nlength (x ∗ y)
      using assms(1) by (simp add: nlength-mono two-times-prod')
  qed
  then show 3 ∗ max (nlength y) (nlength (2 ∗ Arithmetic.prod' x y t)) + 10 + 2 ≤ ttt
    using assms(2) by simp
  let ?xs = canrepr x and ?ys = canrepr y
  let ?r = read (tpsL1 t) ! j1
  have ?r = (⌊x⌋N) (nlength x − t)
    using tpsL1-def jk tapes-at-read'
    by (metis fst-conv length-list-update list-update-swap nth-list-update-eq snd-conv)
  then have r: ?r = canrepr x ! (nlength x − 1 − t)
    using assms contents-def by simp
  have prod' x y (Suc t) = 2 ∗ prod' x y t + (if ?xs ! (length ?xs − 1 − t) = 𝟭 then num ?ys else 0)
    using prod'-def by simp
  also have ... = 2 ∗ prod' x y t + (if ?r = 𝟭 then num ?ys else 0)
    using r by simp
  also have ... = 2 ∗ prod' x y t + (if ?r = 𝟭 then y else 0)
    using canrepr by simp
  finally have prod' x y (Suc t) = 2 ∗ prod' x y t + (if ?r = 𝟭 then y else 0) .
  then show read (tpsL1 t) ! j1 ≠ 𝟭 ⟹ tpsL2 t = tpsL1 t
      and read (tpsL1 t) ! j1 = 𝟭 ⟹
     tpsL2 t = (tpsL1 t) [j3 := (⌊y + 2 ∗ Arithmetic.prod' x y t⌋N, 1)]
    by (simp-all add: add.commute tpsL1-def tpsL2-def)
qed

lemma tmBody1 [transforms-intros]:
  assumes t < nlength x
    and ttt = 17 + 2 ∗ nlength (Arithmetic.prod' x y t) + 3 ∗ nlength (x ∗ y)
  shows transforms tmBody1 (tpsL t) ttt (tpsL2 t)
  unfolding tmBody1-def by (tform tps: jk tpsL-def tpsL1-def assms(1) time: assms(2))
```

lemma *tmBody*:
  assumes $t < nlength\ x$
    and $ttt = 6 + 2 * nlength\ (prod'\ x\ y\ t) + (12 + 3 * nlength\ (x * y))$
  shows *transforms tmBody* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
  unfolding *tmBody-def* **by** (*tform tps*: *jk tpsL-def tpsL2-def assms*(*1*) *time*: *assms*(*2*))

lemma *tmBody′* [*transforms-intros*]:
  assumes $t < nlength\ x$ **and** $ttt = 18 + 5 * nlength\ (x * y)$
  shows *transforms tmBody* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
proof −
  have $6 + 2 * nlength\ (prod'\ x\ y\ t) + (12 + 3 * nlength\ (x * y)) \le 18 + 5 * nlength\ (x * y)$
    using *assms nlength-prod′-le-prod* **by** *simp*
  then show *?thesis*
    using *tmBody assms transforms-monotone* **by** *blast*
qed

lemma *read-contents*:
  fixes *tps* :: *tape list* **and** *j* :: *tapeidx* **and** *zs* :: *symbol list*
  assumes $tps\ !\ j = (\lfloor zs \rfloor, i)$ **and** $i > 0$ **and** $i \le length\ zs$ **and** $j < length\ tps$
  shows $read\ tps\ !\ j = zs\ !\ (i - 1)$
  using *assms tapes-at-read′* **by** *fastforce*

lemma *tmWhile* [*transforms-intros*]:
  assumes $ttt = 1 + 25 * (nlength\ x + nlength\ y) * (nlength\ x + nlength\ y)$
  shows *transforms tmWhile* (*tpsL 0*) *ttt* (*tpsL* (*nlength x*))
  unfolding *tmWhile-def*
proof (*tform*)
  show $read\ (tpsL\ i)\ !\ j1 \neq \triangleright$ **if** $i < nlength\ x$ **for** *i*
  proof −
    have $(tpsL\ i)\ !\ j1 = (\lfloor x \rfloor_N, nlength\ x - i)$
      using *tpsL-def jk* **by** *simp*
    moreover have ∗: $nlength\ x - i > 0$ $nlength\ x - i \le length\ (canrepr\ x)$
      using *that* **by** *simp-all*
    moreover have $length\ (tpsL\ i) = k$
      using *tpsL-def jk* **by** *simp*
    ultimately have $read\ (tpsL\ i)\ !\ j1 = canrepr\ x\ !\ (nlength\ x - i - 1)$
      using *jk read-contents* **by** *simp*
    then show *?thesis*
      using ∗ *bit-symbols-canrepr*
      by (*metis One-nat-def Suc-le-lessD Suc-pred less-numeral-extra*(*4*) *proper-symbols-canrepr*)
  qed
  show $\neg\ read\ (tpsL\ (nlength\ x))\ !\ j1 \neq \triangleright$
  proof −
    have $(tpsL\ (nlength\ x))\ !\ j1 = (\lfloor x \rfloor_N, nlength\ x - nlength\ x)$
      using *tpsL-def jk* **by** *simp*
    then have $(tpsL\ (nlength\ x))\ !\ j1 = (\lfloor x \rfloor_N, 0)$
      by *simp*
    then have $read\ (tpsL\ (nlength\ x))\ !\ j1 = \triangleright$
      using *tapes-at-read′ tpsL-def contents-at-0 jk* **by** (*metis fst-conv length-list-update snd-conv*)
    then show *?thesis*
      by *simp*
  qed
  show $nlength\ x * (18 + 5 * nlength\ (x * y) + 2) + 1 \le ttt$
  proof (*cases* $x = 0$)
    case *True*
    then show *?thesis*
      using *assms* **by** *simp*
  next
    case *False*
    have $nlength\ x * (18 + 5 * nlength\ (x * y) + 2) + 1 = nlength\ x * (20 + 5 * nlength\ (x * y)) + 1$
      by *simp*
    also have ... $\le nlength\ x * (20 + 5 * (nlength\ x + nlength\ y)) + 1$

using *nlength-prod* **by** (*meson add-mono le-refl mult-le-mono*)
also have ... $\leq$ *nlength x* $*$ (*20* $*$ (*nlength x* + *nlength y*) + *5* $*$ (*nlength x* + *nlength y*)) + *1*
proof −
have *1* $\leq$ *nlength x* + *nlength y*
using *False nlength-0* **by** (*simp add*: *Suc-leI*)
then show *?thesis*
by *simp*
qed
also have ... $\leq$ *nlength x* $*$ (*25* $*$ (*nlength x* + *nlength y*)) + *1*
by *simp*
also have ... $\leq$ (*nlength x* + *nlength y*) $*$ (*25* $*$ (*nlength x* + *nlength y*)) + *1*
by *simp*
finally show *?thesis*
using *assms* **by** *linarith*
qed
qed

lemma *tm3*:
assumes *ttt* = *Suc* (*Suc* (*nlength x*)) +
Suc ((*25* $*$ *nlength x* + *25* $*$ *nlength y*) $*$ (*nlength x* + *nlength y*))
shows *transforms tm3 tps0 ttt* (*tpsL* (*nlength x*))
unfolding *tm3-def*
proof (*tform time*: *assms*)
show *tpsL 0* = *tps2*
proof −
have *prod′ x y 0* = *0*
using *prod′-def* **by** *simp*
then show *?thesis*
using *tpsL-def tps2-def jk tps0* **by** (*metis diff-zero list-update-id list-update-swap*)
qed
qed

definition *tps3* ≡ *tps0*
[*j1* := ($\lfloor x \rfloor_N$, *0*),
j3 := ($\lfloor x * y \rfloor_N$, *1*)]

lemma *tm3′* [*transforms-intros*]:
assumes *ttt* = *3* + *26* $*$ (*nlength x* + *nlength y*) $*$ (*nlength x* + *nlength y*)
shows *transforms tm3 tps0 ttt tps3*
proof −
have *Suc* (*Suc* (*nlength x*)) + *Suc* ((*25* $*$ *nlength x* + *25* $*$ *nlength y*) $*$ (*nlength x* + *nlength y*)) $\leq$
Suc (*Suc* (*nlength x* + *nlength y*)) + *Suc* ((*25* $*$ *nlength x* + *25* $*$ *nlength y*) $*$ (*nlength x* + *nlength y*))
by *simp*
also have ... $\leq$ *2* + (*nlength x* + *nlength y*) $*$ (*nlength x* + *nlength y*) + *1* +
25 $*$ (*nlength x* + *nlength y*) $*$ (*nlength x* + *nlength y*)
by (*simp add*: *le-square*)
also have ... = *3* + *26* $*$ (*nlength x* + *nlength y*) $*$ (*nlength x* + *nlength y*)
by *linarith*
finally have *Suc* (*Suc* (*nlength x*)) + *Suc* ((*25* $*$ *nlength x* + *25* $*$ *nlength y*) $*$ (*nlength x* + *nlength y*)) $\leq$
3 + *26* $*$ (*nlength x* + *nlength y*) $*$ (*nlength x* + *nlength y*) **.**
moreover have *tps3* = *tpsL* (*nlength x*)
using *tps3-def tpsL-def* **by** (*simp add*: *prod′*)
ultimately show *?thesis*
using *tm3 assms transforms-monotone* **by** *simp*
qed

definition *tps4* ≡ *tps0*
[*j3* := ($\lfloor x * y \rfloor_N$, *1*)]

lemma *tm4*:
assumes *ttt* = *4* + *26* $*$ (*nlength x* + *nlength y*) $*$ (*nlength x* + *nlength y*)
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*

proof (*tform tps*: *tps3-def jk time*: *assms*)
  show *tps4* = *tps3*[*j1* := *tps3* ! *j1* |+| *1*]
    using *tps4-def tps3-def jk tps0*
    by (*metis One-nat-def add.right-neutral add-Suc-right fst-conv list-update-id list-update-overwrite list-update-swap nth-list-update-eq nth-list-update-neq snd-conv*)
qed

end

end

lemma *transforms-tm-mult* [*transforms-intros*]:
  fixes *j1 j2 j3* :: *tapeidx* **and** *x y k ttt* :: *nat* **and** *tps tps'* :: *tape list*
  assumes *j1* ≠ *j2 j2* ≠ *j3 j3* ≠ *j1 j3* > *0*
  assumes *length tps* = *k*
    and *j1* < *k j2* < *k j3* < *k*
    and *tps* ! *j1* = (⌊*x*⌋$_N$, *1*)
    and *tps* ! *j2* = (⌊*y*⌋$_N$, *1*)
    and *tps* ! *j3* = (⌊*0*⌋$_N$, *1*)
    and *ttt* = *4* + *26* ∗ (*nlength x* + *nlength y*) ∗ (*nlength x* + *nlength y*)
    and *tps'* = *tps* [*j3* := (⌊*x* ∗ *y*⌋$_N$, *1*)]
  shows *transforms* (*tm-mult j1 j2 j3*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-mult j1 j2 j3* **.**
  show *?thesis*
    using *assms loc.tps4-def loc.tm4 loc.tm4-eq-tm-mult* **by** *metis*
qed

2.7.6 Powers

In this section we construct for every $d \in \mathbb{N}$ a Turing machine that computes n^d. The following TMs expect a number n on tape j_1 and output n^d on tape j_3. Another tape, j_2, is used as scratch space to hold intermediate values. The TMs initialize tape j_3 with 1 and then multiply this value by n for d times using the TM *tm-mult*.

fun *tm-pow* :: *nat* ⇒ *tapeidx* ⇒ *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-pow 0 j1 j2 j3 = *tm-setn j3 1* |
  tm-pow (*Suc d*) *j1 j2 j3* =
    tm-pow d j1 j2 j3 ;; (*tm-copyn j3 j2* ;; *tm-setn j3 0* ;; *tm-mult j1 j2 j3* ;; *tm-setn j2 0*)

lemma *tm-pow-tm*:
  assumes *j1* ≠ *j2 j2* ≠ *j3 j3* ≠ *j1*
    and *0* < *j2 0* < *j3 0* < *j1*
  assumes *j1* < *k j2* < *k j3* < *k*
    and *k* ≥ *2*
    and *G* ≥ *4*
  shows *turing-machine k G* (*tm-pow d j1 j2 j3*)
  using *assms tm-copyn-tm tm-setn-tm tm-mult-tm* **by** (*induction d*) *simp-all*

locale *turing-machine-pow* =
  fixes *j1 j2 j3* :: *tapeidx*
begin

definition *tm1* ≡ *tm-copyn j3 j2* ;; *tm-setn j3 0*
definition *tm2* ≡ *tm1* ;; *tm-mult j1 j2 j3*
definition *tm3* ≡ *tm2* ;; *tm-setn j2 0*

fun *tm4* :: *nat* ⇒ *machine* **where**
  tm4 0 = *tm-setn j3 1* |
  tm4 (*Suc d*) = *tm4 d* ;; *tm3*

lemma *tm4-eq-tm-pow*: *tm4 d* = *tm-pow d j1 j2 j3*
  using *tm3-def tm2-def tm1-def* **by** (*induction d*) *simp-all*

context
  fixes *x y k* :: *nat* **and** *tps0* :: *tape list*
  assumes *jk*: $k = length\ tps0\ j1 < k\ j2 < k\ j3 < k$
    $j1 \neq j2\ j2 \neq j3\ j3 \neq j1$
    $0 < j2\ 0 < j3\ 0 < j1$
  assumes *tps0*:
   $tps0\ !\ j1 = (\lfloor x \rfloor_N,\ 1)$
   $tps0\ !\ j2 = (\lfloor 0 \rfloor_N,\ 1)$
   $tps0\ !\ j3 = (\lfloor y \rfloor_N,\ 1)$
begin

definition $tps1 \equiv tps0$
  $[j2 := (\lfloor y \rfloor_N,\ 1),\ j3 := (\lfloor 0 \rfloor_N,\ 1)]$

lemma *tm1* [*transforms-intros*]:
  assumes $ttt = 24 + 5 * nlength\ y$
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def*
proof (*tform tps*: *assms jk tps0 tps1-def*)
  show $ttt = 14 + 3 * (nlength\ y + nlength\ 0) + (10 + 2 * nlength\ y + 2 * nlength\ 0)$
   using *assms* **by** *simp*
qed

definition $tps2 \equiv tps0$
  $[j2 := (\lfloor y \rfloor_N,\ 1),$
   $j3 := (\lfloor x * y \rfloor_N,\ 1)]$

lemma *tm2* [*transforms-intros*]:
  assumes $ttt = 28 + 5 * nlength\ y + (26 * nlength\ x + 26 * nlength\ y) * (nlength\ x + nlength\ y)$
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def*
proof (*tform tps*: *jk tps1-def time*: *assms*)
  show $tps1\ !\ j1 = (\lfloor x \rfloor_N,\ 1)$
   using *jk tps0 tps1-def* **by** *simp*
  show $tps2 = tps1[j3 := (\lfloor x * y \rfloor_N,\ 1)]$
   using *tps2-def tps1-def* **by** *simp*
qed

definition $tps3 \equiv tps0$
  $[j3 := (\lfloor x * y \rfloor_N,\ 1)]$

lemma *tm3*:
  assumes $ttt = 38 + 7 * nlength\ y + (26 * nlength\ x + 26 * nlength\ y) * (nlength\ x + nlength\ y)$
  shows *transforms tm3 tps0 ttt tps3*
  unfolding *tm3-def*
proof (*tform tps*: *jk tps2-def time*: *assms*)
  show $tps3 = tps2[j2 := (\lfloor 0 \rfloor_N,\ 1)]$
   using *tps3-def tps2-def jk* **by** (*metis list-update-id list-update-overwrite list-update-swap tps0*(*2*))
qed

lemma *tm3'*:
  assumes $ttt = 38 + 33 * (nlength\ x + nlength\ y)\ \hat{}\ 2$
  shows *transforms tm3 tps0 ttt tps3*
proof −
  have $38 + 7 * nlength\ y + (26 * nlength\ x + 26 * nlength\ y) * (nlength\ x + nlength\ y) =$
    $38 + 7 * nlength\ y + 26 * (nlength\ x + nlength\ y) * (nlength\ x + nlength\ y)$
   by *simp*
  also have ... $\leq 38 + 33 * (nlength\ x + nlength\ y) * (nlength\ x + nlength\ y)$
  proof −
   have $nlength\ y \leq (nlength\ x + nlength\ y) * (nlength\ x + nlength\ y)$
    by (*meson le-add2 le-square le-trans*)
   then show *?thesis*
    by *linarith*

```
    qed
    also have ... = 38 + 33 ∗ (nlength x + nlength y) ^ 2
      by algebra
    finally have 38 + 7 ∗ nlength y + (26 ∗ nlength x + 26 ∗ nlength y) ∗ (nlength x + nlength y) ≤ ttt
      using assms(1) by simp
    then show ?thesis
      using tm3 transforms-monotone assms by meson
qed

end

lemma tm3'' [transforms-intros]:
  fixes x d k :: nat and tps0 :: tape list
  assumes k = length tps0
    and j1 < k j2 < k j3 < k
  assumes j-neq [simp]: j1 ≠ j2 j2 ≠ j3 j3 ≠ j1
    and j-gt [simp]: 0 < j2 0 < j3 0 < j1
    and tps0 ! j1 = (⌊x⌋N, 1)
    and tps0 ! j2 = (⌊0⌋N, 1)
    and tps0 ! j3 = (⌊x ^ d⌋N, 1)
    and ttt = 71 + 99 ∗ (Suc d) ^ 2 ∗ (nlength x) ^ 2
    and tps' = tps0 [j3 := (⌊x ^ Suc d⌋N, 1)]
  shows transforms tm3 tps0 ttt tps'
proof −
  let ?l = nlength x
  have transforms tm3 tps0 (38 + 33 ∗ (nlength x + nlength (x ^ d)) ^ 2) tps'
    using tm3' assms tps3-def by simp
  moreover have 38 + 33 ∗ (nlength x + nlength (x ^ d)) ^ 2 ≤ 71 + 99 ∗ (Suc d) ^ 2 ∗ ?l ^ 2
  proof −
    have 38 + 33 ∗ (nlength x + nlength (x ^ d)) ^ 2 ≤ 38 + 33 ∗ (Suc (Suc d ∗ ?l)) ^ 2
      using nlength-pow by simp
    also have ... = 38 + 33 ∗ ((Suc d ∗ ?l)^2 + 2 ∗ (Suc d ∗ ?l) ∗ 1 + 1^2)
      by (metis Suc-eq-plus1 add-Suc one-power2 power2-sum)
    also have ... = 38 + 33 ∗ ((Suc d ∗ ?l)^2 + 2 ∗ (Suc d ∗ ?l) + 1)
      by simp
    also have ... ≤ 38 + 33 ∗ ((Suc d ∗ ?l)^2 + 2 ∗ (Suc d ∗ ?l)^2 + 1)
    proof −
      have (Suc d ∗ ?l) ≤ (Suc d ∗ ?l) ^ 2
        by (simp add: le-square power2-eq-square)
      then show ?thesis
        by simp
    qed
    also have ... ≤ 38 + 33 ∗ (3 ∗ (Suc d ∗ ?l)^2 + 1)
      by simp
    also have ... = 38 + 33 ∗ (3 ∗ (Suc d) ^ 2 ∗ ?l^2 + 1)
      by algebra
    also have ... = 71 + 99 ∗ (Suc d) ^ 2 ∗ ?l ^ 2
      by simp
    finally show ?thesis .
  qed
  ultimately show ?thesis
    using transforms-monotone assms(14) by blast
qed

context
  fixes x k :: nat and tps0 :: tape list
  assumes jk: j1 < k j2 < k j3 < k j1 ≠ j2 j2 ≠ j3 j3 ≠ j1 0 < j2 0 < j3 0 < j1 k = length tps0
  assumes tps0:
    tps0 ! j1 = (⌊x⌋N, 1)
    tps0 ! j2 = (⌊0⌋N, 1)
    tps0 ! j3 = (⌊0⌋N, 1)
begin
```

lemma *tm4*:
  fixes d :: *nat*
  assumes $tps' = tps0\ [j3 := (\lfloor x \ \hat{}\ d \rfloor_N,\ 1)]$
    and $ttt = 12 + 71 * d + 99 * d \ \hat{}\ 3 * (nlength\ x)\ \hat{}\ 2$
  shows *transforms* (*tm4 d*) *tps0 ttt tps′*
  using *assms*
proof (*induction d arbitrary*: *tps′ ttt*)
  case *0*
  have $tm4\ 0 = tm\text{-}setn\ j3\ 1$
    by *simp*
  let $?tps = tps0\ [j3 := (\lfloor 1 \rfloor_N,\ 1)]$
  let $?t = 10 + 2 * nlength\ 1$
  have *transforms* (*tm-setn j3 1*) *tps0 ?t ?tps*
    using *transforms-tm-setnI*[*of j3 tps0 0 ?t 1 ?tps*] *jk tps0* **by** *simp*
  then have *transforms* (*tm-setn j3 1*) *tps0 ?t tps′*
    using *0* **by** *simp*
  then show *?case*
    using *0 nlength-1-simp* **by** *simp*
next
  case (*Suc d*)
  note *Suc.IH* [*transforms-intros*]

  let $?l = nlength\ x$
  have *tm4* (*Suc d*) = *tm4 d* ;; *tm3*
    by *simp*
  define *t* **where**
    $t = 12 + 71 * d + 99 * d \ \hat{}\ 3 * (nlength\ x)^2 + (71 + 99 * (Suc\ d)^2 * (nlength\ x)^2)$
  have *transforms* (*tm4 d* ;; *tm3*) *tps0 t tps′*
    by (*tform tps*: *jk tps0 Suc.prems*(*1*) *time*: *t-def*)
  moreover have $t \le 12 + 71 * Suc\ d + 99 * Suc\ d \ \hat{}\ 3 * ?l^2$
  proof −
    have $t = 12 + d * 71 + 99 * d \ \hat{}\ 3 * ?l^2 + (71 + 99 * (Suc\ d)^2 * ?l^2)$
      using *t-def* **by** *simp*
    also have $... = 12 + Suc\ d * 71 + 99 * d \ \hat{}\ 3 * ?l^2 + 99 * (Suc\ d)^2 * ?l^2$
      by *simp*
    also have $... = 12 + Suc\ d * 71 + 99 * ?l\hat{}2 * (d \ \hat{}\ 3 + (Suc\ d)^2)$
      by *algebra*
    also have $... \le 12 + Suc\ d * 71 + 99 * ?l\hat{}2 * Suc\ d \ \hat{}\ 3$
    proof −
      have $Suc\ d \ \hat{}\ 3 = Suc\ d * Suc\ d \ \hat{}\ 2$
        by *algebra*
      also have $... = Suc\ d * (d \ \hat{}\ 2 + 2 * d + 1)$
        by (*metis* (*no-types*, *lifting*) *Suc-1 add.commute add-Suc mult-2 one-power2 plus-1-eq-Suc power2-sum*)
      also have $... = (d + 1) * (d \ \hat{}\ 2 + 2 * d + 1)$
        by *simp*
      also have $... = d \ \hat{}\ 3 + 2 * d \ \hat{}\ 2 + d + d \ \hat{}\ 2 + 2 * d + 1$
        by *algebra*
      also have $... = d \ \hat{}\ 3 + (d + 1) \ \hat{}\ 2 + 2 * d \ \hat{}\ 2 + d$
        by *algebra*
      also have $... \ge d \ \hat{}\ 3 + (d + 1) \ \hat{}\ 2$
        by *simp*
      finally have $Suc\ d \ \hat{}\ 3 \ge d \ \hat{}\ 3 + Suc\ d \ \hat{}\ 2$
        by *simp*
      then show *?thesis*
        by *simp*
    qed
    also have $... = 12 + 71 * Suc\ d + 99 * Suc\ d \ \hat{}\ 3 * ?l\hat{}2$
      by *simp*
    finally show *?thesis* **.**
  qed
  ultimately show *?case*
    using *transforms-monotone Suc* **by** *simp*
qed

end

end

lemma *transforms-tm-pow* [*transforms-intros*]:
  fixes *d* :: *nat*
  assumes $j1 \neq j2$ $j2 \neq j3$ $j3 \neq j1$ $0 < j2$ $0 < j3$ $0 < j1$ $j1 < k$ $j2 < k$ $j3 < k$ $k = length\ tps$
  assumes
    $tps\ !\ j1 = (\lfloor x \rfloor_N,\ 1)$
    $tps\ !\ j2 = (\lfloor 0 \rfloor_N,\ 1)$
    $tps\ !\ j3 = (\lfloor 0 \rfloor_N,\ 1)$
  assumes $ttt = 12 + 71 * d + 99 * d\ \hat{}\ 3 * (nlength\ x)\ \hat{}\ 2$
  assumes $tps' = tps\ [j3 := (\lfloor x\ \hat{}\ d \rfloor_N,\ 1)]$
  shows *transforms* (*tm-pow d j1 j2 j3*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-pow j1 j2 j3* **.**
  show *?thesis*
    using *assms loc.tm4-eq-tm-pow loc.tm4* **by** *metis*
qed

2.7.7 Monomials

A monomial is a power multiplied by a constant coefficient. The following Turing machines have parameters c and d and expect a number x on tape j. They output $c \cdot x^d$ on tape $j + 3$. The tapes $j + 1$ and $j + 2$ are scratch space for use by *tm-pow* and *tm-mult*.

definition *tm-monomial* :: *nat* ⇒ *nat* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-monomial c d j ≡
    tm-pow d j (*j* + *1*) (*j* + *2*) ;;
    tm-setn (*j* + *1*) *c* ;;
    tm-mult (*j* + *1*) (*j* + *2*) (*j* + *3*);;
    tm-setn (*j* + *1*) *0* ;;
    tm-setn (*j* + *2*) *0*

lemma *tm-monomial-tm*:
  assumes $k \geq 2$ **and** $G \geq 4$ **and** $j + 3 < k$ **and** $0 < j$
  shows *turing-machine k G* (*tm-monomial c d j*)
  unfolding *tm-monomial-def*
  using *assms tm-setn-tm tm-mult-tm tm-pow-tm turing-machine-sequential-turing-machine*
  by *simp*

locale *turing-machine-monomial* =
  fixes *c d* :: *nat* **and** *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-pow d j* (*j* + *1*) (*j* + *2*)
definition *tm2* ≡ *tm1* ;; *tm-setn* (*j* + *1*) *c*
definition *tm3* ≡ *tm2* ;; *tm-mult* (*j* + *1*) (*j* + *2*) (*j* + *3*)
definition *tm4* ≡ *tm3* ;; *tm-setn* (*j* + *1*) *0*
definition *tm5* ≡ *tm4* ;; *tm-setn* (*j* + *2*) *0*

lemma *tm5-eq-tm-monomial*: *tm5* = *tm-monomial c d j*
  unfolding *tm1-def tm2-def tm3-def tm4-def tm5-def tm-monomial-def* **by** *simp*

context
  fixes *x k* :: *nat* **and** *tps0* :: *tape list*
  assumes *jk*: $k = length\ tps0$ $j + 3 < k$ $0 < j$
  assumes *tps0*:
    $tps0\ !\ j = (\lfloor x \rfloor_N,\ 1)$
    $tps0\ !\ (j + 1) = (\lfloor 0 \rfloor_N,\ 1)$
    $tps0\ !\ (j + 2) = (\lfloor 0 \rfloor_N,\ 1)$
    $tps0\ !\ (j + 3) = (\lfloor 0 \rfloor_N,\ 1)$
begin

definition *tps1* ≡ *tps0* [(*j* + *2*) := (⌊*x* ^ *d*⌋$_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
  assumes *ttt* = *12* + *71* ∗ *d* + *99* ∗ *d* ^ *3* ∗ (*nlength x*) ^ *2*
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def* **by** (*tform tps*: *assms tps0 jk tps1-def*)

definition *tps2* ≡ *tps0*
  [*j* + *2* := (⌊*x* ^ *d*⌋$_N$, *1*),
   j + *1* := (⌊*c*⌋$_N$, *1*)]

lemma *tm2* [*transforms-intros*]:
  assumes *ttt* = *22* + *71* ∗ *d* + *99* ∗ *d* ^ *3* ∗ (*nlength x*)^2 + *2* ∗ *nlength c*
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def*
proof (*tform tps*: *assms tps0 jk tps2-def tps1-def*)
  show *ttt* = *12* + *71* ∗ *d* + *99* ∗ *d* ^ *3* ∗ (*nlength x*)^2 + (*10* + *2* ∗ *nlength 0* + *2* ∗ *nlength c*)
    using *assms*(*1*) **by** *simp*
qed

definition *tps3* ≡ *tps0*
  [*j* + *2* := (⌊*x* ^ *d*⌋$_N$, *1*),
   j + *1* := (⌊*c*⌋$_N$, *1*),
   j + *3* := (⌊*c* ∗ *x* ^ *d*⌋$_N$, *1*)]

lemma *tm3* [*transforms-intros*]:
  assumes *ttt* = *26* + *71* ∗ *d* + *99* ∗ *d* ^ *3* ∗ (*nlength x*)^2 + *2* ∗ *nlength c* +
    26 ∗ (*nlength c* + *nlength* (*x* ^ *d*)) ^ *2*
  shows *transforms tm3 tps0 ttt tps3*
  unfolding *tm3-def*
proof (*tform tps*: *tps2-def tps3-def tps0 jk*)
  show *ttt* = *22* + *71* ∗ *d* + *99* ∗ *d* ^ *3* ∗ (*nlength x*)^2 + *2* ∗ *nlength c* +
      (*4* + *26* ∗ (*nlength c* + *nlength* (*x* ^ *d*)) ∗ (*nlength c* + *nlength* (*x* ^ *d*)))
    using *assms* **by** *algebra*
qed

definition *tps4* ≡ *tps0*
  [*j* + *2* := (⌊*x* ^ *d*⌋$_N$, *1*),
   j + *3* := (⌊*c* ∗ *x* ^ *d*⌋$_N$, *1*)]

lemma *tm4* [*transforms-intros*]:
  assumes *ttt* = *36* + *71* ∗ *d* + *99* ∗ *d* ^ *3* ∗ (*nlength x*)^2 + *4* ∗ *nlength c* +
    26 ∗ (*nlength c* + *nlength* (*x* ^ *d*))^2
  shows *transforms tm4 tps0 ttt tps4*
  unfolding *tm4-def*
proof (*tform tps*: *tps4-def tps3-def tps0 jk time*: *assms*)
  show *tps4* = *tps3*[*j* + *1* := (⌊*0*⌋$_N$, *1*)]
    unfolding *tps4-def tps3-def*
    using *jk tps0*(*2*) *list-update-id*[*of tps0 Suc j*] **by** (*simp add*: *list-update-swap*)
qed

definition *tps5* ≡ *tps0*
  [*j* + *3* := (⌊*c* ∗ *x* ^ *d*⌋$_N$, *1*)]

lemma *tm5*:
  assumes *ttt* = *46* + *71* ∗ *d* + *99* ∗ *d* ^ *3* ∗ (*nlength x*)^2 + *4* ∗ *nlength c* +
    26 ∗ (*nlength c* + *nlength* (*x* ^ *d*))^2 +
    (*2* ∗ *nlength* (*x* ^ *d*))
  shows *transforms tm5 tps0 ttt tps5*
  unfolding *tm5-def*
proof (*tform tps*: *tps5-def tps4-def jk time*: *assms*)
  show *tps5* = *tps4*[*j* + *2* := (⌊*0*⌋$_N$, *1*)]

```
    unfolding tps5-def tps4-def
    using jk tps0 list-update-id[of tps0 Suc (Suc j)]
    by (simp add: list-update-swap)
qed

lemma tm5':
  assumes ttt = 46 + 71 * d + 99 * d ^ 3 * (nlength x)² + 32 * (nlength c + nlength (x ^ d))²
  shows transforms tm5 tps0 ttt tps5
proof –
  let ?t = 46 + 71 * d + 99 * d ^ 3 * (nlength x)² + 4 * nlength c +
    26 * (nlength c + nlength (x ^ d))² + (2 * nlength (x ^ d))
  have ?t ≤ 46 + 71 * d + 99 * d ^ 3 * (nlength x)² + 4 * nlength c +
    28 * (nlength c + nlength (x ^ d))²
  proof –
    have 2 * nlength (x ^ d) ≤ 2 * (nlength c + nlength (x ^ d))²
      by (meson add-leE eq-imp-le mult-le-mono2 power2-nat-le-imp-le)
    then show ?thesis
      by simp
  qed
  also have ... ≤ 46 + 71 * d + 99 * d ^ 3 * (nlength x)² + 32 * (nlength c + nlength (x ^ d))²
  proof –
    have 4 * nlength c ≤ 4 * (nlength c + nlength (x ^ d))²
      by (simp add: power2-nat-le-eq-le power2-nat-le-imp-le)
    then show ?thesis
      by simp
  qed
  also have ... = ttt
    using assms(1) by simp
  finally have ?t ≤ ttt .
  then show ?thesis
    using assms transforms-monotone tm5 by blast
qed

end

end

lemma transforms-tm-monomialI [transforms-intros]:
  fixes ttt x k :: nat and tps tps' :: tape list and j :: tapeidx
  assumes j > 0 and j + 3 < k and k = length tps
  assumes
    tps ! j = (⌊x⌋N, 1)
    tps ! (j + 1) = (⌊0⌋N, 1)
    tps ! (j + 2) = (⌊0⌋N, 1)
    tps ! (j + 3) = (⌊0⌋N, 1)
  assumes ttt = 46 + 71 * d + 99 * d ^ 3 * (nlength x)² + 32 * (nlength c + nlength (x ^ d))²
  assumes tps' = tps[j + 3 := (⌊c * x ^ d⌋N, 1)]
  shows transforms (tm-monomial c d j) tps ttt tps'
proof –
  interpret loc: turing-machine-monomial c d j .
  show ?thesis
    using loc.tm5-eq-tm-monomial loc.tm5' loc.tps5-def assms by simp
qed
```

2.7.8 Polynomials

A polynomial is a sum of monomials. In this section we construct for every polynomial function p a Turing machine that on input $x \in \mathbb{N}$ outputs $p(x)$.

According to our definition of polynomials (see Section 2.1.4), we can represent each polynomial by a list of coefficients. The value of such a polynomial with coefficient list *cs* on input x is given by the next function. In the following definition, the coefficients of the polynomial are in reverse order, which simplifies the Turing machine later.

```
definition polyvalue :: nat list ⇒ nat ⇒ nat where
  polyvalue cs x ≡ (∑ i←[0..<length cs]. rev cs ! i * x ^ i)

lemma polyvalue-Nil: polyvalue [] x = 0
  using polyvalue-def by simp

lemma sum-upt-snoc: (∑ i←[0..<length (zs @ [z])]. (zs @ [z]) ! i * x ^ i) =
    (∑ i←[0..<length zs]. zs ! i * x ^ i) + z * x ^ (length zs)
 by simp (smt (verit, ccfv-SIG) length-map less-diff-conv map-equality-iff map-nth nth-append nth-upt zero-less-diff)

lemma polyvalue-Cons: polyvalue (c # cs) x = c * x ^ (length cs) + polyvalue cs x
proof −
  have polyvalue (c # cs) x = (∑ i←[0..<Suc (length cs)]. (rev cs @ [c]) ! i * x ^ i)
    using polyvalue-def by simp
  also have ... = (∑ i←[0..<length (rev cs @ [c])]. (rev cs @ [c]) ! i * x ^ i)
    by simp
  also have ... = (∑ i←[0..<length (rev cs)]. (rev cs) ! i * x ^ i) + c * x ^ (length (rev cs))
    using sum-upt-snoc by blast
  also have ... = (∑ i←[0..<length cs]. (rev cs) ! i * x ^ i) + c * x ^ (length cs)
    by simp
  finally show ?thesis
    using polyvalue-def by simp
qed

lemma polyvalue-Cons-ge: polyvalue (c # cs) x ≥ polyvalue cs x
  using polyvalue-Cons by simp

lemma polyvalue-Cons-ge2: polyvalue (c # cs) x ≥ c * x ^ (length cs)
  using polyvalue-Cons by simp

lemma sum-list-const: (∑ -←ns. c) = c * length ns
  using sum-list-triv[of c ns] by simp

lemma polyvalue-le: polyvalue cs x ≤ Max (set cs) * length cs * Suc x ^ length cs
proof −
  define cmax where cmax = Max (set (rev cs))
  have polyvalue cs x = (∑ i←[0..<length cs]. rev cs ! i * x ^ i)
    using polyvalue-def by simp
  also have ... ≤ (∑ i←[0..<length cs]. cmax * x ^ i)
  proof −
    have rev cs ! i ≤ cmax if i < length cs for i
      using that cmax-def by (metis List.finite-set Max-ge length-rev nth-mem)
    then show ?thesis
      by (metis (no-types, lifting) atLeastLessThan-iff mult-le-mono1 set-upt sum-list-mono)
  qed
  also have ... = cmax * (∑ i←[0..<length cs]. x ^ i)
    using sum-list-const-mult by blast
  also have ... ≤ cmax * (∑ i←[0..<length cs]. Suc x ^ i)
    by (simp add: power-mono sum-list-mono)
  also have ... ≤ cmax * (∑ i←[0..<length cs]. Suc x ^ length cs)
  proof −
    have Suc x ^ i ≤ Suc x ^ length cs if i < length cs for i
      using that by (simp add: dual-order.strict-implies-order pow-mono)
    then show ?thesis
      by (metis atLeastLessThan-iff mult-le-mono2 set-upt sum-list-mono)
  qed
  also have ... = cmax * length cs * Suc x ^ length cs
    using sum-list-const[of - [0..<length cs]] by simp
  finally have polyvalue cs x ≤ cmax * length cs * Suc x ^ length cs .
  moreover have cmax = Max (set cs)
    using cmax-def by simp
  ultimately show ?thesis
    by simp
```

```
qed

lemma nlength-polyvalue:
 nlength (polyvalue cs x) ≤ nlength (Max (set cs)) + nlength (length cs) + Suc (length cs ∗ nlength (Suc x))
proof −
  have nlength (polyvalue cs x) ≤ nlength (Max (set cs) ∗ length cs ∗ Suc x ^ length cs)
    using polyvalue-le nlength-mono by simp
  also have ... ≤ nlength (Max (set cs) ∗ length cs) + nlength (Suc x ^ length cs)
    using nlength-prod by simp
  also have ... ≤ nlength (Max (set cs)) + nlength(length cs) + Suc (length cs ∗ nlength (Suc x))
    by (meson add-mono nlength-pow nlength-prod)
  finally show ?thesis .
qed
```

The following Turing machines compute polynomials given as lists of coefficients. If the polynomial is given by coefficients *cs*, the TM *tm-polycoef cs j* expect a number n on tape j and writes $p(n)$ to tape $j+4$. The tapes $j+1$, $j+2$, and $j+3$ are auxiliary tapes for use by *tm-monomial*.

```
fun tm-polycoef :: nat list ⇒ tapeidx ⇒ machine where
  tm-polycoef [] j = [] |
  tm-polycoef (c # cs) j =
     tm-polycoef cs j ;;
     (tm-monomial c (length cs) j ;;
      tm-add (j + 3) (j + 4) ;;
      tm-setn (j + 3) 0)

lemma tm-polycoef-tm:
  assumes k ≥ 2 and G ≥ 4 and j + 4 < k and 0 < j
  shows turing-machine k G (tm-polycoef cs j)
proof (induction cs)
  case Nil
  then show ?case
    by (simp add: assms(1) assms(2) turing-machine-def)
next
  case (Cons c cs)
  moreover have
    turing-machine k G (tm-monomial c (length cs) j ;; tm-add (j + 3) (j + 4) ;; tm-setn (j + 3) 0)
    using tm-monomial-tm tm-add-tm tm-setn-tm assms
    by simp
  ultimately show ?case
    by simp
qed

locale turing-machine-polycoef =
  fixes j :: tapeidx
begin

definition tm1 c cs ≡ tm-monomial c (length cs) j
definition tm2 c cs ≡ tm1 c cs ;; tm-add (j + 3) (j + 4)
definition tm3 c cs ≡ tm2 c cs ;; tm-setn (j + 3) 0

fun tm4 :: nat list ⇒ machine where
  tm4 [] = [] |
  tm4 (c # cs) = tm4 cs ;; tm3 c cs

lemma tm4-eq-tm-polycoef: tm4 zs = tm-polycoef zs j
proof (induction zs)
  case Nil
  then show ?case
    by simp
next
  case (Cons z zs)
  then show ?case
    by (simp add: tm1-def tm2-def tm3-def)
```

qed

context
  fixes *x y k* :: *nat* **and** *tps0* :: *tape list*
  fixes *c* :: *nat* **and** *cs* :: *nat list*
  assumes *jk*: $0 < j$ $j + 4 < k$ $k = length\ tps0$
  assumes *tps0*:
    $tps0 ! j = (\lfloor x \rfloor_N, 1)$
    $tps0 ! (j + 1) = (\lfloor 0 \rfloor_N, 1)$
    $tps0 ! (j + 2) = (\lfloor 0 \rfloor_N, 1)$
    $tps0 ! (j + 3) = (\lfloor 0 \rfloor_N, 1)$
    $tps0 ! (j + 4) = (\lfloor y \rfloor_N, 1)$
begin

abbreviation $d \equiv length\ cs$

definition $tps1 \equiv tps0$
  $[j + 3 := (\lfloor c * x \,\hat{}\, (length\ cs) \rfloor_N, 1)]$

lemma *tm1* [*transforms-intros*]:
  assumes $ttt = 46 + 71 * d + 99 * d \,\hat{}\, 3 * (nlength\ x)^2 +$
      $32 * (nlength\ c + nlength\ (x \,\hat{}\, d))^2$
  shows *transforms* (*tm1 c cs*) *tps0 ttt tps1*
  unfolding *tm1-def* **by** (*tform tps*: *assms jk tps0 tps1-def*)

definition $tps2 = tps0$
  $[j + 3 := (\lfloor c * x \,\hat{}\, (length\ cs) \rfloor_N, 1),$
   $j + 4 := (\lfloor c * x \,\hat{}\, (length\ cs) + y \rfloor_N, 1)]$

lemma *tm2* [*transforms-intros*]:
  assumes $ttt = 46 + 71 * d + 99 * d \,\hat{}\, 3 * (nlength\ x)^2 +$
    $32 * (nlength\ c + nlength\ (x \,\hat{}\, d))^2 +$
    $(3 * max\ (nlength\ (c * x \,\hat{}\, d))\ (nlength\ y) + 10)$
  shows *transforms* (*tm2 c cs*) *tps0 ttt tps2*
  unfolding *tm2-def* **by** (*tform tps*: *tps1-def tps2-def jk tps0 time*: *assms*)

definition $tps3 \equiv tps0$
  $[j + 4 := (\lfloor c * x \,\hat{}\, d + y \rfloor_N, 1)]$

lemma *tm3*:
  assumes $ttt = 66 + 71 * d + 99 * d \,\hat{}\, 3 * (nlength\ x)^2 +$
      $32 * (nlength\ c + nlength\ (x \,\hat{}\, d))^2 +$
      $3 * max\ (nlength\ (c * x \,\hat{}\, d))\ (nlength\ y) +$
      $2 * nlength\ (c * x \,\hat{}\, d)$
  shows *transforms* (*tm3 c cs*) *tps0 ttt tps3*
  unfolding *tm3-def*
proof (*tform tps*: *tps2-def tps3-def jk tps0 time*: *assms*)
  show $tps3 = tps2[j + 3 := (\lfloor 0 \rfloor_N, 1)]$
    using *tps3-def tps2-def jk tps0*
    by (*smt* (*verit*) *One-nat-def add-2-eq-Suc add-left-cancel lessI less-numeral-extra*(*4*) *list-update-id list-update-overwrite list-update-swap numeral-3-eq-3 numeral-Bit0 plus-1-eq-Suc*)
qed

definition $tps3' \equiv tps0$
  $[j + 4 := (\lfloor c * x \,\hat{}\, length\ cs + y \rfloor_N, 1)]$

lemma *tm3'*:
  assumes $ttt = 66 + 71 * d + 99 * d \,\hat{}\, 3 * (nlength\ x)^2 +$
      $32 * (nlength\ c + nlength\ (x \,\hat{}\, d))^2 +$
      $5 * max\ (nlength\ (c * x \,\hat{}\, d))\ (nlength\ y)$
  shows *transforms* (*tm3 c cs*) *tps0 ttt tps3'*
proof −
  have $66 + 71 * d + 99 * d \,\hat{}\, 3 * (nlength\ x)^2 +$

32 ∗ (*nlength c* + *nlength* (*x* ^ *d*))^2 +
3 ∗ *max* (*nlength* (*c* ∗ *x* ^ *d*)) (*nlength y*) +
2 ∗ *nlength* (*c* ∗ *x* ^ *d*) ≤
66 + *71* ∗ *d* + *99* ∗ *d* ^ *3* ∗ (*nlength x*)^2 +
32 ∗ (*nlength c* + *nlength* (*x* ^ *d*))^2 +
3 ∗ *max* (*nlength* (*c* ∗ *x* ^ *d*)) (*nlength y*) +
2 ∗ *max* (*nlength* (*c* ∗ *x* ^ *d*)) (*nlength y*)
by *simp*
also have ... = *66* + *71* ∗ *d* + *99* ∗ *d* ^ *3* ∗ (*nlength x*)^2 +
32 ∗ (*nlength c* + *nlength* (*x* ^ *d*))^2 +
5 ∗ *max* (*nlength* (*c* ∗ *x* ^ *d*)) (*nlength y*)
by *simp*
finally have *66* + *71* ∗ *d* + *99* ∗ *d* ^ *3* ∗ (*nlength x*)^2 +
32 ∗ (*nlength c* + *nlength* (*x* ^ *d*))^2 +
3 ∗ *max* (*nlength* (*c* ∗ *x* ^ *d*)) (*nlength y*) +
2 ∗ *nlength* (*c* ∗ *x* ^ *d*) ≤ *ttt*
using *assms*(*1*) **by** *simp*
moreover have *tps3′* = *tps3*
using *tps3′-def tps3-def* **by** *simp*
ultimately show *?thesis*
using *tm3 transforms-monotone* **by** *simp*
qed

end

lemma *tm3′′* [*transforms-intros*]:
fixes *c* :: *nat* **and** *cs* :: *nat list*
fixes *x k* :: *nat* **and** *tps0 tps′* :: *tape list*
assumes *k* = *length tps0* **and** *j* + *4* < *k* **and** *0* < *j*
assumes
tps0 ! *j* = (⌊*x*⌋$_N$, *1*)
tps0 ! (*j* + *1*) = (⌊*0*⌋$_N$, *1*)
tps0 ! (*j* + *2*) = (⌊*0*⌋$_N$, *1*)
tps0 ! (*j* + *3*) = (⌊*0*⌋$_N$, *1*)
tps0 ! (*j* + *4*) = (⌊*polyvalue cs x*⌋$_N$, *1*)
assumes *ttt* = *66* +
71 ∗ (*length cs*) +
99 ∗ (*length cs*) ^ *3* ∗ (*nlength x*)^2 +
32 ∗ (*nlength c* + *nlength* (*x* ^ (*length cs*)))^2 +
5 ∗ *max* (*nlength* (*c* ∗ *x* ^ (*length cs*))) (*nlength* (*polyvalue cs x*))
assumes *tps′* = *tps0*
[*j* + *4* := (⌊*polyvalue* (*c* # *cs*) *x*⌋$_N$, *1*)]
shows *transforms* (*tm3 c cs*) *tps0 ttt tps′*
using *assms tm3′*[**where** *?y*=*polyvalue cs x*] *tps3′-def polyvalue-Cons* **by** *simp*

lemma *pow-le-pow-Suc*:
fixes *a b* :: *nat*
shows *a* ^ *b* ≤ *Suc a* ^ *Suc b*
proof −
have *a* ^ *b* ≤ *Suc a* ^ *b*
by (*simp add*: *power-mono*)
then show *?thesis*
by *simp*
qed

lemma *tm4*:
fixes *x k* :: *nat* **and** *tps0* :: *tape list*
fixes *cs* :: *nat list*
assumes *k* = *length tps0* **and** *j* + *4* < *k* **and** *0* < *j*
assumes
tps0 ! *j* = (⌊*x*⌋$_N$, *1*)
tps0 ! (*j* + *1*) = (⌊*0*⌋$_N$, *1*)
tps0 ! (*j* + *2*) = (⌊*0*⌋$_N$, *1*)

```
    tps0 ! (j + 3) = (⌊0⌋N, 1)
    tps0 ! (j + 4) = (⌊0⌋N, 1)
  assumes ttt: ttt = length cs *
     (66 +
      71 * (length cs) +
      99 * (length cs) ^ 3 * (nlength x)^2 +
      32 * (Max (set (map nlength cs)) + nlength (Suc x ^ length cs))^2 +
      5 * nlength (polyvalue cs x))
  shows transforms (tm4 cs) tps0 ttt (tps0[j + 4 := (⌊polyvalue cs x⌋N, 1)])
  using ttt
proof (induction cs arbitrary: ttt)
  case Nil
  then show ?case
    using polyvalue-Nil transforms-Nil assms by (metis list.size(3) list-update-id mult-is-0 tm4.simps(1))
next
  case (Cons c cs)
  note Cons.IH [transforms-intros]

  have tm4def: tm4 (c # cs) = tm4 cs ;; tm3 c cs
    by simp

  let ?t1 = d cs *
    (66 + 71 * d cs + 99 * d cs ^ 3 * (nlength x)^2 +
     32 * (Max (nlength ‘ set cs) + nlength (Suc x ^ d cs))^2 +
     5 * nlength (polyvalue cs x))
  let ?t2 = 66 + 71 * d cs + 99 * d cs ^ 3 * (nlength x)^2 +
     32 * (nlength c + nlength (x ^ d cs))^2 +
     5 * max (nlength (c * x ^ d cs)) (nlength (polyvalue cs x))
  define t where t = ?t1 + ?t2
  have tm4: transforms (tm4 (c # cs)) tps0 t (tps0[j + 4 := (⌊polyvalue (c # cs) x⌋N, 1)])
    unfolding tm4def by (tform tps: assms t-def)

  have ?t1 ≤ d cs *
    (66 + 71 * d (c#cs) + 99 * d cs ^ 3 * (nlength x)^2 +
     32 * (Max (nlength ‘ set cs) + nlength (Suc x ^d cs))^2 +
     5 * nlength (polyvalue cs x))
    by simp
  also have ... ≤ d cs *
    (66 + 71 * d (c#cs) + 99 * d (c#cs) ^ 3 * (nlength x)^2 +
     32 * (Max (nlength ‘ set cs) + nlength (Suc x ^d cs))^2 +
     5 * nlength (polyvalue cs x))
    by simp
  also have ... ≤ d cs *
    (66 + 71 * d (c#cs) + 99 * d (c#cs) ^ 3 * (nlength x)^2 +
     32 * (Max (nlength ‘ set (c#cs)) + nlength (Suc x ^d cs))^2 +
     5 * nlength (polyvalue cs x))
    by simp
  also have ... ≤ d cs *
    (66 + 71 * d (c#cs) + 99 * d (c#cs) ^ 3 * (nlength x)^2 +
     32 * (Max (nlength ‘ set (c#cs)) + nlength (Suc x ^d (c#cs)))^2 +
     5 * nlength (polyvalue cs x))
    using nlength-mono by simp
  also have ... ≤ d cs *
    (66 + 71 * d (c#cs) + 99 * d (c#cs) ^ 3 * (nlength x)^2 +
     32 * (Max (nlength ‘ set (c#cs)) + nlength (Suc x ^d (c#cs)))^2 +
     5 * nlength (polyvalue (c#cs) x))
    using nlength-mono polyvalue-Cons-ge by simp
  finally have t1: ?t1 ≤ d cs *
    (66 + 71 * d (c#cs) + 99 * d (c#cs) ^ 3 * (nlength x)^2 +
     32 * (Max (nlength ‘ set (c#cs)) + nlength (Suc x ^d (c#cs)))^2 +
     5 * nlength (polyvalue (c#cs) x))
    (is ?t1 ≤ d cs * ?t3) .
```

have *?t2* $\leq$
66 + *71* $*$ *d* (*c* # *cs*) + *99* $*$ *d cs* ^ *3* $*$ (*nlength x*)^2 +
32 $*$ (*nlength c* + *nlength* (*x* ^ *d cs*))^2 +
5 $*$ *max* (*nlength* (*c* $*$ *x* ^ *d cs*)) (*nlength* (*polyvalue cs x*))
by *simp*
also have ... $\leq$ *66* + *71* $*$ *d* (*c* # *cs*) + *99* $*$ *d* (*c* # *cs*) ^ *3* $*$ (*nlength x*)^2 +
32 $*$ (*nlength c* + *nlength* (*x* ^ *d cs*))^2 +
5 $*$ *max* (*nlength* (*c* $*$ *x* ^ *d cs*)) (*nlength* (*polyvalue cs x*))
by *simp*
also have ... $\leq$ *66* + *71* $*$ *d* (*c* # *cs*) + *99* $*$ *d* (*c* # *cs*) ^ *3* $*$ (*nlength x*)^2 +
32 $*$ (*Max* (*set* (*map nlength* (*c* # *cs*))) + *nlength* (*x* ^ *d cs*))^2 +
5 $*$ *max* (*nlength* (*c* $*$ *x* ^ *d cs*)) (*nlength* (*polyvalue cs x*))
by *simp*
also have ... $\leq$ *66* + *71* $*$ *d* (*c* # *cs*) + *99* $*$ *d* (*c* # *cs*) ^ *3* $*$ (*nlength x*)^2 +
32 $*$ (*Max* (*set* (*map nlength* (*c* # *cs*))) + *nlength* (*Suc x* ^ *d* (*c*#*cs*)))^2 +
5 $*$ *max* (*nlength* (*c* $*$ *x* ^ *d cs*)) (*nlength* (*polyvalue cs x*))
using *nlength-mono pow-le-pow-Suc* **by** *simp*
also have ... $\leq$ *66* + *71* $*$ *d* (*c* # *cs*) + *99* $*$ *d* (*c* # *cs*) ^ *3* $*$ (*nlength x*)^2 +
32 $*$ (*Max* (*set* (*map nlength* (*c* # *cs*))) + *nlength* (*Suc x* ^ *d* (*c*#*cs*)))^2 +
5 $*$ *max* (*nlength* (*c* $*$ *x* ^ *d cs*)) (*nlength* (*polyvalue* (*c*#*cs*) *x*))
proof −
have *nlength* (*polyvalue cs x*) $\leq$ *nlength* (*polyvalue* (*c*#*cs*) *x*)
using *polyvalue-Cons* **by** (*simp add*: *nlength-mono*)
then show *?thesis*
by *simp*
qed
also have ... $\leq$ *66* + *71* $*$ *d* (*c* # *cs*) + *99* $*$ *d* (*c* # *cs*) ^ *3* $*$ (*nlength x*)^2 +
32 $*$ (*Max* (*set* (*map nlength* (*c* # *cs*))) + *nlength* (*Suc x* ^ *d* (*c*#*cs*)))^2 +
5 $*$ *max* (*nlength* (*polyvalue* (*c*#*cs*) *x*)) (*nlength* (*polyvalue* (*c*#*cs*) *x*))
using *nlength-mono polyvalue-Cons-ge2* **by** *simp*
also have ... $\leq$ *66* + *71* $*$ *d* (*c* # *cs*) + *99* $*$ *d* (*c* # *cs*) ^ *3* $*$ (*nlength x*)^2 +
32 $*$ (*Max* (*set* (*map nlength* (*c* # *cs*))) + *nlength* (*Suc x* ^ *d* (*c*#*cs*)))^2 +
5 $*$ *nlength* (*polyvalue* (*c*#*cs*) *x*)
by *simp*
finally have *t2*: *?t2* $\leq$ *?t3*
by *simp*

have *t* $\leq$ *d cs* $*$ *?t3* + *?t3*
using *t1 t2 t-def add-le-mono* **by** *blast*
then have *t* $\leq$ *d* (*c*#*cs*) $*$ *?t3*
by *simp*
moreover have *ttt* = *d* (*c*#*cs*) $*$ *?t3*
using *Cons* **by** *simp*
ultimately have *t* $\leq$ *ttt*
by *simp*
then show *?case*
using *tm4 transforms-monotone* **by** *simp*
qed

end

The time bound in the previous lemma for *tm-polycoef* is a bit unwieldy. It depends not only on the length of the input x but also on the list of coefficients of the polynomial p and on the value $p(x)$. Next we bound this time bound by a simpler expression of the form $d + d \cdot |x|^2$ where d depends only on the polynomial. This is accomplished by the next three lemmas.

lemma *tm-polycoef-time-1*: $\exists$ *d*. $\forall$ *x*. *nlength* (*polyvalue cs x*) $\leq$ *d* + *d* $*$ *nlength x*
proof −
{ fix *x*
have *nlength* (*polyvalue cs x*) $\leq$ *nlength* (*Max* (*set cs*)) + *nlength* (*length cs*) + *Suc* (*length cs* $*$ *nlength* (*Suc x*))
using *nlength-polyvalue* **by** *simp*
also have ... = *nlength* (*Max* (*set cs*)) + *nlength* (*length cs*) + *1* + *length cs* $*$ *nlength* (*Suc x*)
(**is** - = *?a* + *length cs* $*$ *nlength* (*Suc x*))

by *simp*
also have ... $\leq$ *?a* + *length cs* $*$ (*Suc* (*nlength x*))
using *nlength-Suc* **by** (*meson add-mono-thms-linordered-semiring*(*2*) *mult-le-mono2*)
also have ... = *?a* + *length cs* + *length cs* $*$ *nlength x*
(**is** - = *?b* + *length cs* $*$ *nlength x*)
by *simp*
also have ... $\leq$ *?b* + *?b* $*$ *nlength x*
by (*meson add-left-mono le-add2 mult-le-mono1*)
finally have *nlength* (*polyvalue cs x*) $\leq$ *?b* + *?b* $*$ *nlength x* **.**
}
then show *?thesis*
by *blast*
qed

lemma *tm-polycoef-time-2*: $\exists$ *d*. $\forall$ *x*. (*Max* (*set* (*map nlength cs*)) + *nlength* (*Suc x* ^ *length cs*))^2 $\leq$ *d* + *d* $*$ *nlength x* ^ *2*
proof −
{ fix *x*
have (*Max* (*set* (*map nlength cs*)) + *nlength* (*Suc x* ^ *length cs*))^2 $\leq$
(*Max* (*set* (*map nlength cs*)) + *Suc* (*nlength* (*Suc x*) $*$ *length cs*))^2
using *nlength-pow* **by** (*simp add*: *mult.commute*)
also have ... = (*Suc* (*Max* (*set* (*map nlength cs*))) + *nlength* (*Suc x*) $*$ *length cs*)^2
(**is** - = (*?a* + *?b*)^*2*)
by *simp*
also have ... = *?a* ^ *2* + *2* $*$ *?a* $*$ *?b* + *?b* ^ *2*
by *algebra*
also have ... $\leq$ *?a* ^ *2* + *2* $*$ *?a* $*$ *?b* ^ *2* + *?b* ^ *2*
by (*meson add-le-mono dual-order.eq-iff mult-le-mono2 power2-nat-le-imp-le*)
also have ... $\leq$ *?a* ^ *2* + (*2* $*$ *?a* + *1*) $*$ *?b* ^ *2*
by *simp*
also have ... = *?a* ^ *2* + (*2* $*$ *?a* + *1*) $*$ (*length cs*) ^ *2* $*$ *nlength* (*Suc x*) ^ *2*
by *algebra*
also have ... $\leq$ *?a* ^ *2* + (*2* $*$ *?a* + *1*) $*$ (*length cs*) ^ *2* $*$ *Suc* (*nlength x*) ^ *2*
using *nlength-Suc* **by** *simp*
also have ... = *?a* ^ *2* + (*2* $*$ *?a* + *1*) $*$ (*length cs*) ^ *2* $*$ (*nlength x* ^ *2* + *2* $*$ *nlength x* + *1*)
by (*smt* (*verit*) *Suc-eq-plus1 add.assoc mult-2 nat-1-add-1 one-power2 plus-1-eq-Suc power2-sum*)
also have ... $\leq$ *?a* ^ *2* + (*2* $*$ *?a* + *1*) $*$ (*length cs*) ^ *2* $*$ (*nlength x* ^ *2* + *2* $*$ *nlength x* ^ *2* + *1*)
proof −
have *nlength x* ^ *2* + *2* $*$ *nlength x* + *1* $\leq$ *nlength x* ^ *2* + *2* $*$ *nlength x* ^ *2* + *1*
by (*metis add-le-mono1 add-mono-thms-linordered-semiring*(*2*) *le-square mult.commute mult-le-mono1 numerals*(*1*) *power-add-numeral power-one-right semiring-norm*(*2*))
then show *?thesis*
by *simp*
qed
also have ... = *?a* ^ *2* + (*2* $*$ *?a* + *1*) $*$ (*length cs*) ^ *2* $*$ (*3* $*$ *nlength x* ^ *2* + *1*)
by *simp*
also have ... = *?a* ^ *2* + (*2* $*$ *?a* + *1*) $*$ (*length cs*) ^ *2* + (*2* $*$ *?a* + *1*) $*$ (*length cs*) ^ *2* $*$ *3* $*$ *nlength x* ^ *2*
(**is** - = - + *?c* $*$ *nlength x* ^ *2*)
by *simp*
also have ... $\leq$ *?a* ^ *2* + *?c* + *?c* $*$ *nlength x* ^ *2*
(**is** - $\leq$ *?d* + *?c* $*$ *nlength x* ^ *2*)
by *simp*
also have ... $\leq$ *?d* + *?d* $*$ *nlength x* ^ *2*
by *simp*
finally have (*Max* (*set* (*map nlength cs*)) + *nlength* (*Suc x* ^ *length cs*))^2 $\leq$ *?d* + *?d* $*$ *nlength x* ^ *2* **.**
}
then show *?thesis*
by *auto*
qed

lemma *tm-polycoef-time-3*:
$\exists$ *d*. $\forall$ *x*. *length cs* $*$
(*66* +

```
    71 * length cs +
    99 * length cs ^ 3 * (nlength x)^2 +
    32 * (Max (set (map nlength cs)) + nlength (Suc x ^ length cs))^2 +
    5 * nlength (polyvalue cs x)) ≤ d + d * nlength x ^ 2
proof -
  obtain d1 where d1: ∀x. nlength (polyvalue cs x) ≤ d1 + d1 * nlength x
    using tm-polycoef-time-1 by auto
  obtain d2 where d2: ∀x. (Max (set (map nlength cs)) + nlength (Suc x ^ length cs))^2 ≤ d2 + d2 * nlength
x ^ 2
    using tm-polycoef-time-2 by auto
  { fix x
    let ?lhs =  length cs *
      (66 +
      71 * length cs +
      99 * length cs ^ 3 * (nlength x)^2 +
      32 * (Max (set (map nlength cs)) + nlength (Suc x ^ length cs))^2 +
      5 * nlength (polyvalue cs x))
    let ?n = nlength x
    have ?lhs ≤ length cs *
        (66 + 71 * length cs + 99 * length cs ^ 3 * ?n ^ 2 +
        32 * (d2 + d2 * ?n ^ 2) + 5 * (d1 + d1 * ?n))
      using d1 d2 add-le-mono mult-le-mono2 nat-add-left-cancel-le by presburger
    also have ... ≤ length cs *
        (66 + 71 * length cs + 99 * length cs ^ 3 * ?n ^ 2 +
        32 * (d2 + d2 * ?n ^ 2) + 5 * (d1 + d1 * ?n ^ 2))
      by (simp add: le-square power2-eq-square)
    also have ... = length cs *
        (66 + 71 * length cs + 99 * length cs ^ 3 * ?n ^ 2 +
        32 * d2 + 32 * d2 * ?n ^ 2 + 5 * d1 + 5 * d1 * ?n ^ 2)
      by simp
    also have ... = length cs *
        (66 + 71 * length cs + 32 * d2 + 5 * d1 +
        (99 * length cs ^ 3 + 32 * d2 + 5 * d1) * ?n ^ 2)
      by algebra
    also have ... = length cs * (66 + 71 * length cs + 32 * d2 + 5 * d1) +
        length cs * (99 * length cs ^ 3 + 32 * d2 + 5 * d1) * ?n ^ 2
          (is - = ?a + ?b * ?n ^ 2)
      by algebra
    also have ... ≤ max ?a ?b + max ?a ?b * ?n ^ 2
      by (simp add: add-mono-thms-linordered-semiring(1))
    finally have ?lhs ≤ max ?a ?b + max ?a ?b * ?n ^ 2 .
  }
  then show ?thesis
    by auto
qed
```

According to our definition of *polynomial* (see Section 2.1.4) every polynomial has a list of coefficients. Therefore the next definition is well-defined for polynomials p.

```
definition coefficients :: (nat ⇒ nat) ⇒ nat list where
  coefficients p ≡ SOME cs. ∀n. p n = (∑i←[0..<length cs]. cs ! i * n ^ i)
```

The d in our upper bound of the form $d + d \cdot |x|^2$ for the running time of *tm-polycoef* depends on the polynomial. It is given by the next function:

```
definition d-polynomial :: (nat ⇒ nat) ⇒ nat where
  d-polynomial p ≡
   (let cs = rev (coefficients p)
    in SOME d. ∀x. length cs *
    (66 +
     71 * length cs +
     99 * length cs ^ 3 * (nlength x)^2 +
     32 * (Max (set (map nlength cs)) + nlength (Suc x ^ length cs))^2 +
     5 * nlength (polyvalue cs x)) ≤ d + d * nlength x ^ 2)
```

The Turing machine *tm-polycoef* has the coefficients of a polynomial as parameter. Next we devise a similar Turing machine that has the polynomial, as a function $\mathbb{N} \to \mathbb{N}$, as parameter.

definition *tm-polynomial* :: (*nat* ⇒ *nat*) ⇒ *tapeidx* ⇒ *machine* **where**
  tm-polynomial p j ≡ *tm-polycoef* (*rev* (*coefficients p*)) *j*

lemma *tm-polynomial-tm*:
  assumes $k \geq 2$ **and** $G \geq 4$ **and** $0 < j$ **and** $j + 4 < k$
  shows *turing-machine k G* (*tm-polynomial p j*)
  using *assms tm-polynomial-def tm-polycoef-tm* **by** *simp*

lemma *transforms-tm-polynomialI* [*transforms-intros*]:
  fixes *p* :: *nat* ⇒ *nat* **and** *j* :: *tapeidx*
  fixes *k x* :: *nat* **and** *tps tps'* :: *tape list*
  assumes $0 < j$ **and** $k = length\ tps$ **and** $j + 4 < k$
    and *polynomial p*
  assumes
    $tps\ !\ j = (\lfloor x \rfloor_N,\ 1)$
    $tps\ !\ (j + 1) = (\lfloor 0 \rfloor_N,\ 1)$
    $tps\ !\ (j + 2) = (\lfloor 0 \rfloor_N,\ 1)$
    $tps\ !\ (j + 3) = (\lfloor 0 \rfloor_N,\ 1)$
    $tps\ !\ (j + 4) = (\lfloor 0 \rfloor_N,\ 1)$
  assumes $ttt = d\text{-}polynomial\ p + d\text{-}polynomial\ p * nlength\ x\ \hat{}\ 2$
  assumes $tps' = tps$
    $[j + 4 := (\lfloor p\ x \rfloor_N,\ 1)]$
  shows *transforms* (*tm-polynomial p j*) *tps ttt tps'*
proof −
  let $?P = \lambda x.\ \forall n.\ p\ n = (\sum i{\leftarrow}[0..{<}length\ x].\ x\ !\ i * n\ \hat{}\ i)$
  define *cs* **where** *cs* = (*SOME x. ?P x*)
  moreover have *ex*: ∃ *cs. ?P cs*
    using *assms*(*4*) *polynomial-def* **by** *simp*
  ultimately have *?P cs*
    using *someI-ex*[*of ?P*] **by** *blast*
  then have *1*: *polyvalue* (*rev cs*) *x* = *p x*
    using *polyvalue-def* **by** *simp*

  let *?cs* = *rev cs*
  have *d-polynomial p* = (*SOME d*. ∀ *x*. *length ?cs* ∗
    (*66* +
     71 ∗ *length ?cs* +
     99 ∗ *length ?cs* ^ *3* ∗ $(nlength\ x)^2$ +
     32 ∗ (*Max* (*set* (*map nlength ?cs*)) + *nlength* (*Suc x* ^ *length ?cs*))^2 +
     5 ∗ *nlength* (*polyvalue ?cs x*)) ≤ *d* + *d* ∗ *nlength x* ^ *2*)
    using *cs-def coefficients-def d-polynomial-def* **by** *simp*
  then have ∗: ∀ *x*. *length ?cs* ∗
    (*66* +
     71 ∗ *length ?cs* +
     99 ∗ *length ?cs* ^ *3* ∗ $(nlength\ x)^2$ +
     32 ∗ (*Max* (*set* (*map nlength ?cs*)) + *nlength* (*Suc x* ^ *length ?cs*))^2 +
     5 ∗ *nlength* (*polyvalue ?cs x*)) ≤ (*d-polynomial p*) + (*d-polynomial p*) ∗ *nlength x* ^ *2*
    using *tm-polycoef-time-3 someI-ex*[*OF tm-polycoef-time-3*] **by** *presburger*

  let *?ttt* = *length ?cs* ∗
    (*66* +
     71 ∗ *length ?cs* +
     99 ∗ *length ?cs* ^ *3* ∗ $(nlength\ x)^2$ +
     32 ∗ (*Max* (*set* (*map nlength ?cs*)) + *nlength* (*Suc x* ^ *length ?cs*))^2 +
     5 ∗ *nlength* (*polyvalue ?cs x*))

  interpret *loc*: *turing-machine-polycoef j* **.**

  have *transforms* (*loc.tm4 ?cs*) *tps ?ttt* (*tps*[$j + 4 := (\lfloor polyvalue\ ?cs\ x \rfloor_N,\ 1)$])
    using *loc.tm4 assms* ∗ **by** *blast*
  then have *transforms* (*loc.tm4 ?cs*) *tps ?ttt* (*tps*[$j + 4 := (\lfloor p\ x \rfloor_N,\ 1)$])

using *1* **by** *simp*
then have *transforms* (*loc.tm4 ?cs*) *tps ?ttt tps'*
using *assms*(*11*) **by** *simp*
moreover have *loc.tm4 ?cs* = *tm-polynomial p j*
using *tm-polynomial-def loc.tm4-eq-tm-polycoef coefficients-def cs-def* **by** *simp*
ultimately have *transforms* (*tm-polynomial p j*) *tps ?ttt tps'*
by *simp*
then show *transforms* (*tm-polynomial p j*) *tps ttt tps'*
using ∗ *assms*(*10*) *transforms-monotone* **by** *simp*
qed

2.7.9 Division by two

In order to divide a number by two, a Turing machine can shift all symbols on the tape containing the number to the left, of course without overwriting the start symbol.

The next command implements the left shift. It scans the tape j from right to left and memorizes the current symbol on the last tape. It works very similar to *cmd-double* only in the opposite direction. Upon reaching the start symbol, it moves the head one cell to the right.

definition *cmd-halve* :: *tapeidx* ⇒ *command* **where**
cmd-halve j rs ≡
(*if rs* ! *j* = *1 then 1 else 0*,
(*map* (λ*i*.
if i = *j then*
if rs ! *j* = ▷ *then* (*rs* ! *i*, *Right*)
else if last rs = ▷ *then* (□, *Left*)
else (*tosym* (*todigit* (*last rs*)), *Left*)
else if i = *length rs* − *1 then* (*tosym* (*todigit* (*rs* ! *j*)), *Stay*)
else (*rs* ! *i*, *Stay*)) [*0*..<*length rs*]))

lemma *turing-command-halve*:
assumes *G* ≥ *4* **and** *0* < *j* **and** *j* < *k*
shows *turing-command* (*Suc k*) *1 G* (*cmd-halve j*)
proof
show ⋀*gs*. *length gs* = *Suc k* ⟹ *length* ([!!] *cmd-halve j gs*) = *length gs*
using *cmd-halve-def* **by** *simp*
moreover have *0* ≠ *Suc k* − *1*
using *assms* **by** *simp*
ultimately show ⋀*gs*. *length gs* = *Suc k* ⟹ *0* < *Suc k* ⟹ *cmd-halve j gs* [.] *0* = *gs* ! *0*
using *assms cmd-halve-def* **by** (*smt* (*verit*) *One-nat-def ab-semigroup-add-class.add-ac*(*1*) *diff-Suc-1 length-map neq0-conv nth-map nth-upt plus-1-eq-Suc prod.sel*(*1*) *prod.sel*(*2*))
show *cmd-halve j gs* [.] *j'* < *G*
if *length gs* = *Suc k* (⋀*i*. *i* < *length gs* ⟹ *gs* ! *i* < *G*) *j'* < *length gs*
for *gs j'*
proof −
have *cmd-halve j gs* [!] *j'* =
(*if j'* = *j then*
if gs ! *j* = ▷ *then* (*gs* ! *j'*, *Right*)
else if last gs = ▷ *then* (□, *Left*)
else (*tosym* (*todigit* (*last gs*)), *Left*)
else if j' = *length gs* − *1 then* (*tosym* (*todigit* (*gs* ! *j*)), *Stay*)
else (*gs* ! *j'*, *Stay*))
using *cmd-halve-def that*(*3*) **by** *simp*
moreover consider *j'* = *j* | *j'* = *k* | *j'* ≠ *j* ∧ *j'* ≠ *k*
by *auto*
ultimately show *?thesis*
using *that assms* **by** (*cases*) *simp-all*
qed
show ⋀*gs*. *length gs* = *Suc k* ⟹ [∗] (*cmd-halve j gs*) ≤ *1*
using *cmd-halve-def* **by** *simp*
qed

lemma *sem-cmd-halve-2*:

```
  assumes j < k
    and bit-symbols xs
    and length tps = Suc k
    and i ≤ length xs
    and i > 0
    and z = 0 ∨ z = 1
    and tps ! j = (⌊xs⌋, i)
    and tps ! k = ⌈z⌉
    and tps' = tps[j := tps ! j |:=| z |−| 1, k := ⌈xs ! (i − 1)⌉]
  shows sem (cmd-halve j) (0, tps) = (0, tps')
proof (rule semI)
  show proper-command (Suc k) (cmd-halve j)
    using cmd-halve-def by simp
  show length tps = Suc k length tps' = Suc k
    using assms(3,9) by simp-all
  define rs where rs = read tps
  then have lenrs: length rs = Suc k
    using assms(3) read-length by simp
  have rsj: rs ! j = xs ! (i − 1)
    using rs-def assms tapes-at-read' contents-inbounds
    by (metis fst-conv le-imp-less-Suc less-imp-le-nat snd-conv)
  then have rsj': rs ! j > 1
    using assms Suc-1 Suc-diff-1 Suc-le-lessD by (metis eval-nat-numeral(3) less-Suc-eq)
  then show fst (cmd-halve j (read tps)) = 0
    using cmd-halve-def rs-def by simp
  have lastrs: last rs = z
    using assms rs-def onesie-read tapes-at-read'
    by (metis diff-Suc-1 last-conv-nth length-0-conv lenrs lessI nat.simps(3))
  show act (cmd-halve j (read tps) [!] j') (tps ! j') = tps' ! j' if j' < Suc k for j'
  proof −
    have j' < length rs
      using that lenrs by simp
    then have ∗: cmd-halve j rs [!] j' =
      (if j' = j then
         if rs ! j = ▷ then (rs ! j', Right)
         else if last rs = ▷ then (□, Left)
         else (tosym (todigit (last rs)), Left)
       else if j' = length rs − 1 then (tosym (todigit (rs ! j)), Stay)
       else (rs ! j', Stay))
      using cmd-halve-def by simp
    consider j' = j | j' = k | j' ≠ j ∧ j' ≠ k
      by auto
    then show ?thesis
    proof (cases)
      case 1
      then have cmd-halve j (read tps) [!] j' = (tosym (todigit (last rs)), Left)
        using rs-def rsj' lastrs ∗ assms(6) by auto
      then have cmd-halve j (read tps) [!] j' = (z, Left)
        using lastrs assms(6) by auto
      moreover have tps' ! j' = tps ! j |:=| z |−| 1
        using 1 assms(1,3,9) by simp
      ultimately show ?thesis
        using act-Left' 1 that rs-def by metis
    next
      case 2
      then have cmd-halve j (read tps) [!] j' = (tosym (todigit (rs ! j)), Stay)
        using rs-def ∗ lenrs assms(1) by simp
      moreover have tps' ! j' = ⌈xs ! (i − 1)⌉
        using assms 2 by simp
      moreover have tps ! j' = ⌈z⌉
        using assms 2 by simp
      moreover have tosym (todigit (rs ! j)) = xs ! (i − 1)
      proof −
```

```
        have xs ! (i − 1) = 0 ∨ xs ! (i − 1) = 1
          using rsj rs-def assms by simp
        then show ?thesis
          using One-nat-def add-2-eq-Suc′ numeral-3-eq-3 rsj by presburger
      qed
      ultimately show ?thesis
        using act-onesie by simp
    next
      case 3
      then show ?thesis
        using ∗ act-Stay that assms lenrs rs-def by simp
    qed
  qed
qed

lemma sem-cmd-halve-1:
  assumes j < k
    and bit-symbols xs
    and length tps = Suc k
    and 0 < length xs
    and tps ! j = (⌊xs⌋, length xs)
    and tps ! k = ⌈▷⌉
    and tps′ = tps[j := tps ! j |:=| □ |−| 1, k := ⌈xs ! (length xs − 1)⌉]
  shows sem (cmd-halve j) (0, tps) = (0, tps′)
proof (rule semI)
  show proper-command (Suc k) (cmd-halve j)
    using cmd-halve-def by simp
  show length tps = Suc k length tps′ = Suc k
    using assms(3,7) by simp-all
  define rs where rs = read tps
  then have lenrs: length rs = Suc k
    using assms(3) read-length by simp
  have rsj: rs ! j = xs ! (length xs − 1)
    using rs-def assms tapes-at-read′ contents-inbounds
    by (metis One-nat-def fst-conv le-eq-less-or-eq le-imp-less-Suc snd-conv)
  then have rsj′: rs ! j > 1
    using assms(2,4) by (metis One-nat-def Suc-1 diff-less lessI less-add-Suc2 numeral-3-eq-3 plus-1-eq-Suc)
  then show fst (cmd-halve j (read tps)) = □
    using cmd-halve-def rs-def by simp
  have lastrs: last rs = ▷
    using assms rs-def onesie-read tapes-at-read′
    by (metis diff-Suc-1 last-conv-nth length-0-conv lenrs lessI nat.simps(3))
  show act (cmd-halve j (read tps) [!] j′) (tps ! j′) = tps′ ! j′ if j′ < Suc k for j′
  proof −
    have j′ < length rs
      using that lenrs by simp
    then have ∗: cmd-halve j rs [!] j′ =
      (if j′ = j then
          if rs ! j = ▷ then (rs ! j′, Right)
          else if last rs = ▷ then (□, Left)
          else (tosym (todigit (last rs)), Left)
        else if j′ = length rs − 1 then (tosym (todigit (rs ! j)), Stay)
        else (rs ! j′, Stay))
      using cmd-halve-def by simp
    consider j′ = j | j′ = k | j′ ≠ j ∧ j′ ≠ k
      by auto
    then show ?thesis
    proof (cases)
      case 1
      then have cmd-halve j (read tps) [!] j′ = (□, Left)
        using rs-def rsj′ lastrs ∗ by simp
      then show ?thesis
        using act-Left′ 1 that rs-def assms(1,3,7) by simp
```

```
    next
      case 2
      then have cmd-halve j (read tps) [!] j' = (tosym (todigit (rs ! j)), Stay)
        using rs-def * lenrs assms(1) by simp
      moreover have tps' ! j' = ⌈xs ! (length xs − 1)⌉
        using assms 2 by simp
      moreover have tps ! j' = ⌈▷⌉
        using assms 2 by simp
      ultimately show ?thesis
        using act-onesie assms 2 that rs-def rsj
        by (smt (verit) One-nat-def Suc-1 add-2-eq-Suc' diff-less numeral-3-eq-3 zero-less-one)
    next
      case 3
      then show ?thesis
        using * act-Stay that assms lenrs rs-def by simp
    qed
  qed
qed

lemma sem-cmd-halve-0:
  assumes j < k
    and length tps = Suc k
    and tps ! j = (⌊xs⌋, 0)
    and tps ! k = ⌈z⌉
    and tps' = tps[j := tps ! j |+| 1, k := ⌈𝟬⌉]
  shows sem (cmd-halve j) (0, tps) = (1, tps')
proof (rule semI)
  show proper-command (Suc k) (cmd-halve j)
    using cmd-halve-def by simp
  show length tps = Suc k length tps' = Suc k
    using assms(2,5) by simp-all
  show fst (cmd-halve j (read tps)) = 1
    using cmd-halve-def assms contents-at-0 tapes-at-read'
    by (smt (verit) fst-conv le-eq-less-or-eq not-less not-less-eq snd-conv)
  show act (cmd-halve j (read tps) [!] j') (tps ! j') = tps' ! j' if j' < Suc k for j'
  proof −
    define gs where gs = read tps
    then have length gs = Suc k
      using assms by (simp add: read-length)
    then have j' < length gs
      using that by simp
    then have *: cmd-halve j gs [!] j' =
      (if j' = j then
         if gs ! j = ▷ then (gs ! j', Right)
         else if last gs = ▷ then (□, Left)
         else (tosym (todigit (last gs)), Left)
       else if j' = length gs − 1 then (tosym (todigit (gs ! j)), Stay)
       else (gs ! j', Stay))
      using cmd-halve-def by simp
    have gsj: gs ! j = ▷
      using gs-def assms(1,2,3) by (metis contents-at-0 fstI less-Suc-eq sndI tapes-at-read')
    consider j' = j | j' = k | j' ≠ j ∧ j' ≠ k
      by auto
    then show ?thesis
    proof (cases)
      case 1
      then have cmd-halve j (read tps) [!] j' = (gs ! j', Right)
        using gs-def gsj * by simp
      then show ?thesis
        using act-Right assms 1 that gs-def by (metis length-list-update lessI nat-neq-iff nth-list-update)
    next
      case 2
      then have cmd-halve j (read tps) [!] j' = (tosym (todigit (gs ! j)), Stay)
```

using *gs-def* $*$ ‹*length gs* = *Suc k*› *assms*(*1*) **by** *simp*
moreover have *tps*′ ! *j*′ = ⌈**0**⌉
using *assms 2* **by** *simp*
moreover have *tps* ! *j*′ = ⌈*z*⌉
using *assms 2* **by** *simp*
ultimately show *?thesis*
using *act-onesie assms 2 that gs-def gsj*
by (*smt* (*verit*, *best*) *One-nat-def Suc-1 add-2-eq-Suc*′ *less-Suc-eq-0-disj less-numeral-extra*(*3*) *nat.inject numeral-3-eq-3*)
next
case *3*
then show *?thesis*
using $*$ *act-Stay that assms*(*2*,*5*) ‹*length gs* = *Suc k*› *gs-def* **by** *simp*
qed
qed
qed

definition *tm-halve* :: *tapeidx* ⇒ *machine* **where**
tm-halve j ≡ [*cmd-halve j*]

lemma *tm-halve-tm*:
assumes *k* ≥ *2* **and** *G* ≥ *4* **and** *0* < *j* **and** *j* < *k*
shows *turing-machine* (*Suc k*) *G* (*tm-halve j*)
using *tm-halve-def turing-command-halve assms* **by** *auto*

lemma *exe-cmd-halve-0*:
assumes *j* < *k*
and *length tps* = *Suc k*
and *tps* ! *j* = (⌊*xs*⌋, *0*)
and *tps* ! *k* = ⌈*z*⌉
and *tps*′ = *tps*[*j* := *tps* ! *j* |+| *1*, *k* := ⌈**0**⌉]
shows *exe* (*tm-halve j*) (*0*, *tps*) = (*1*, *tps*′)
using *assms sem-cmd-halve-0 tm-halve-def exe-lt-length* **by** *simp*

lemma *execute-cmd-halve-0*:
assumes *j* < *k*
and *length tps* = *Suc k*
and *tps* ! *j* = (⌊[]⌋, *0*)
and *tps* ! *k* = ⌈▷⌉
and *tps*′ = *tps*[*j* := *tps* ! *j* |+| *1*, *k* := ⌈**0**⌉]
shows *execute* (*tm-halve j*) (*0*, *tps*) *1* = (*1*, *tps*′)
using *tm-halve-def exe-lt-length sem-cmd-halve-0 assms* **by** *simp*

definition *shift* :: *tape* ⇒ *nat* ⇒ *tape* **where**
shift tp y ≡ (λ*x*. *if x* ≤ *y then* (*fst tp*) *x else* (*fst tp*) (*Suc x*), *y*)

lemma *shift-update*: *y* > *0* ⟹ *shift tp y* |:=| (*fst tp*) (*Suc y*) |−| *1* = *shift tp* (*y* − *1*)
unfolding *shift-def* **by** *fastforce*

lemma *shift-contents-0*:
assumes *length xs* > *0*
shows *shift* (⌊*xs*⌋, *length xs*) *0* = (⌊*tl xs*⌋, *0*)
proof −
have *shift* (⌊*xs*⌋, *length xs*) *0* = (⌊*drop 1 xs*⌋, *0*)
using *shift-def contents-def* **by** *fastforce*
then show *?thesis*
by (*simp add*: *drop-Suc*)
qed

lemma *proper-bit-symbols*: *bit-symbols ws* ⟹ *proper-symbols ws*
by *auto*

lemma *bit-symbols-shift*:

assumes $t < length\ ws$ **and** *bit-symbols ws*
shows $|.|\ (shift\ (\lfloor ws \rfloor,\ length\ ws)\ (length\ ws - t)) \neq 1$
using *assms shift-def contents-def nat-neq-iff proper-bit-symbols* **by** *simp*

lemma *exe-cmd-halve-1*:
assumes $j < k$
and $length\ tps = Suc\ k$
and *bit-symbols xs*
and $length\ xs > 0$
and $tps\ !\ j = (\lfloor xs \rfloor,\ length\ xs)$
and $tps\ !\ k = \lceil \triangleright \rceil$
and $tps' = tps[j := tps\ !\ j\ |{:=}|\ \Box\ |{-}|\ 1,\ k := \lceil xs\ !\ (length\ xs - 1) \rceil]$
shows $exe\ (tm\text{-}halve\ j)\ (0,\ tps) = (0,\ tps')$
using *tm-halve-def exe-lt-length sem-cmd-halve-1 assms* **by** *simp*

lemma *shift-contents-eq-take-drop*:
assumes $length\ xs > 0$
and $ys = take\ i\ xs\ @\ drop\ (Suc\ i)\ xs$
and $i > 0$
and $i < length\ xs$
shows $shift\ (\lfloor xs \rfloor,\ length\ xs)\ i = (\lfloor ys \rfloor,\ i)$
proof $-$
have $shift\ (\lfloor xs \rfloor,\ length\ xs)\ i = (\lambda x.\ if\ x \leq i\ then\ \lfloor xs \rfloor\ x\ else\ \lfloor xs \rfloor\ (Suc\ x),\ i)$
using *shift-def* **by** *auto*
moreover have $(\lambda x.\ if\ x \leq i\ then\ \lfloor xs \rfloor\ x\ else\ \lfloor xs \rfloor\ (Suc\ x)) = \lfloor take\ i\ xs\ @\ drop\ (Suc\ i)\ xs \rfloor$
(**is** $?l = ?r$)
proof
fix x
consider $x = 0\ |\ 0 < x \wedge x \leq i\ |\ i < x \wedge x \leq length\ xs - 1\ |\ length\ xs - 1 < x$
by *linarith*
then show $?l\ x = ?r\ x$
proof (*cases*)
case *1*
then show *?thesis*
using *assms contents-def* **by** *simp*
next
case *2*
then have $?l\ x = \lfloor xs \rfloor\ x$
by *simp*
then have *lhs*: $?l\ x = xs\ !\ (x - 1)$
using *assms 2* **by** *simp*
have $?r\ x = (take\ i\ xs\ @\ drop\ (Suc\ i)\ xs)\ !\ (x - 1)$
using *assms 2* **by** *auto*
then have $?r\ x = xs\ !\ (x - 1)$
using *assms(4) 2*
by (*metis diff-less le-eq-less-or-eq length-take less-trans min-absorb2 nth-append nth-take zero-less-one*)
then show *?thesis*
using *lhs* **by** *simp*
next
case *3*
then have $?l\ x = \lfloor xs \rfloor\ (Suc\ x)$
by *simp*
then have *lhs*: $?l\ x = xs\ !\ x$
using *3 assms* **by** *auto*
have $?r\ x = (take\ i\ xs\ @\ drop\ (Suc\ i)\ xs)\ !\ (x - 1)$
using *assms 3* **by** *auto*
then have $?r\ x = drop\ (Suc\ i)\ xs\ !\ (x - 1 - i)$
using *assms(3,4) 3*
by (*smt* (*verit*) *Suc-diff-1 dual-order.strict-trans length-take less-Suc-eq min-absorb2 nat-less-le nth-append*)
then have $?r\ x = xs\ !\ x$
using *assms 3* **by** *simp*
then show *?thesis*
using *lhs* **by** *simp*

```
    next
      case 4
      then show ?thesis
        using contents-def by auto
    qed
  qed
  ultimately show ?thesis
    using assms(2) by simp
qed

lemma exe-cmd-halve-2:
  assumes j < k
    and bit-symbols xs
    and length tps = Suc k
    and i ≤ length xs
    and i > 0
    and z = 𝟬 ∨ z = 𝟭
    and tps ! j = (⌊xs⌋, i)
    and tps ! k = ⌈z⌉
    and tps' = tps[j := tps ! j |:=| z |-| 1, k := ⌈xs ! (i − 1)⌉]
  shows exe (tm-halve j) (0, tps) = (0, tps')
  using tm-halve-def exe-lt-length sem-cmd-halve-2 assms by simp

lemma shift-contents-length-minus-1:
  assumes length xs > 0
  shows shift (⌊xs⌋, length xs) (length xs − 1) = (⌊xs⌋, length xs) |:=| □ |-| 1
  using contents-def shift-def assms by fastforce

lemma execute-tm-halve-1-less:
  assumes j < k
    and length tps = Suc k
    and bit-symbols xs
    and length xs > 0
    and tps ! j = (⌊xs⌋, length xs)
    and tps ! k = ⌈▷⌉
    and t ≥ 1
    and t ≤ length xs
  shows execute (tm-halve j) (0, tps) t = (0, tps
      [j := shift (tps ! j) (length xs − t),
       k := ⌈xs ! (length xs − t)⌉])
  using assms(7,8)
proof (induction t rule: nat-induct-at-least)
  case base
  have execute (tm-halve j) (0, tps) 1 = exe (tm-halve j) (0, tps)
    by simp
  also have ... = (0, tps[j := tps ! j |:=| □ |-| 1, k := ⌈xs ! (length xs − 1)⌉])
    using assms exe-cmd-halve-1 by simp
  also have ... = (0, tps[j := shift (tps ! j) (length xs − 1), k := ⌈xs ! (length xs − 1)⌉])
    using shift-contents-length-minus-1 assms(4,5) by simp
  finally show ?case .
next
  case (Suc t)
  then have t < length xs
    by simp
  let ?ys = take (length xs − t) xs @ drop (Suc (length xs − t)) xs
  have execute (tm-halve j) (0, tps) (Suc t) = exe (tm-halve j) (execute (tm-halve j) (0, tps) t)
    by simp
  also have ... = exe (tm-halve j) (0, tps
      [j := shift (tps ! j) (length xs − t),
       k := ⌈xs ! (length xs − t)⌉])
    using Suc by simp
  also have ... = exe (tm-halve j) (0, tps
      [j := shift (⌊xs⌋, length xs) (length xs − t),
```

$k := \lceil xs ! (length\ xs - t)\rceil])$
using *assms*(*5*) **by** *simp*
also have ... = *exe* (*tm-halve j*) (*0*, *tps*
[$j := (\lfloor ?ys\rfloor, length\ xs - t)$,
$k := \lceil xs ! (length\ xs - t)\rceil]$)
(**is** - = *exe* - (*0*, *?tps*))
using *shift-contents-eq-take-drop Suc assms* **by** *simp*
also have ... = (*0*, *?tps*
[$j := ?tps ! j$ |:=| $(xs ! (length\ xs - t))$ |−| *1*,
$k := \lceil ?ys ! (length\ xs - t - 1)\rceil]$)
proof −
let $?i = length\ xs - t$
let $?z = xs ! ?i$
have *1*: *bit-symbols ?ys*
using *assms*(*3*) **by** (*intro bit-symbols-append*) *simp-all*
have *2*: *length ?tps = Suc k*
using *assms*(*2*) **by** *simp*
have *3*: $?i \le length\ ?ys$
using *Suc assms* **by** *simp*
have *4*: $?i > 0$
using *Suc assms* **by** *simp*
have *5*: $?z = 2 \vee ?z = 3$
using *assms*(*3*,*4*) *Suc* **by** *simp*
have *6*: $?tps ! j = (\lfloor ?ys\rfloor, ?i)$
using *assms*(*1*,*2*) **by** *simp*
have *7*: $?tps ! k = \lceil ?z\rceil$
using *assms*(*2*) **by** *simp*
then show *?thesis*
using *exe-cmd-halve-2*[*OF assms*(*1*) *1 2 3 4 5 6 7*] **by** *simp*
qed
also have ... = (*0*, *tps*
[$j := ?tps ! j$ |:=| $(xs ! (length\ xs - t))$ |−| *1*,
$k := \lceil ?ys ! (length\ xs - t - 1)\rceil]$)
using *assms* **by** (*smt* (*verit*) *list-update-overwrite list-update-swap*)
also have ... = (*0*, *tps*
[$j := (\lfloor ?ys\rfloor, length\ xs - t)$ |:=| $(xs ! (length\ xs - t))$ |−| *1*,
$k := \lceil ?ys ! (length\ xs - t - 1)\rceil]$)
using *assms*(*1*,*2*) **by** *simp*
also have ... = (*0*, *tps*
[$j := shift\ (\lfloor xs\rfloor, length\ xs)\ (length\ xs - Suc\ t)$,
$k := \lceil xs ! (length\ xs - (Suc\ t))\rceil]$)
proof −
have $(\lfloor ?ys\rfloor, length\ xs - t)$ |:=| $xs ! (length\ xs - t)$ |−| *1* =
$shift\ (\lfloor xs\rfloor, length\ xs)\ (length\ xs - t)$ |:=| $(xs ! (length\ xs - t))$ |−| *1*
using *shift-contents-eq-take-drop One-nat-def Suc Suc-le-lessD* ‹$t < length\ xs$› *assms*(*4*) *diff-less zero-less-diff*
by *presburger*
also have ... = $shift\ (\lfloor xs\rfloor, length\ xs)\ (length\ xs - Suc\ t)$
using *shift-update*[*of* $length\ xs - t$ $(\lfloor xs\rfloor, length\ xs)$] *assms Suc* **by** *simp*
finally have $(\lfloor ?ys\rfloor, length\ xs - t)$ |:=| $xs ! (length\ xs - t)$ |−| *1* =
$shift\ (\lfloor xs\rfloor, length\ xs)\ (length\ xs - Suc\ t)$ **.**
moreover have $?ys ! (length\ xs - t - 1) = xs ! (length\ xs - Suc\ t)$
using *Suc assms* ‹$t < length\ xs$›
by (*metis* (*no-types*, *lifting*) *diff-Suc-eq-diff-pred diff-Suc-less diff-commute diff-less length-take min-less-iff-conj nth-append nth-take zero-less-diff zero-less-one*)
ultimately show *?thesis*
by *simp*
qed
also have ... = (*0*, *tps*
[$j := shift\ (tps ! j)\ (length\ xs - (Suc\ t))$,
$k := \lceil xs ! (length\ xs - (Suc\ t))\rceil]$)
using *assms*(*5*) **by** *simp*
finally show *?case* **.**
qed

```
lemma execute-tm-halve-1:
  assumes j < k
    and length tps = Suc k
    and bit-symbols xs
    and length xs > 0
    and tps ! j = (⌊xs⌋, length xs)
    and tps ! k = ⌈▷⌉
    and tps' = tps[j := (⌊tl xs⌋, 1), k := ⌈𝟬⌉]
  shows execute (tm-halve j) (0, tps) (Suc (length xs)) = (1, tps')
proof –
  have execute (tm-halve j) (0, tps) (length xs) = (0, tps[j := shift (tps ! j) 0, k := ⌈xs ! 0⌉])
    using execute-tm-halve-1-less[OF assms(1−6), where ?t=length xs] assms(4) by simp
  also have ... = (0, tps[j := shift (⌊xs⌋, length xs) 0, k := ⌈xs ! 0⌉])
    using assms(5) by simp
  also have ... = (0, tps[j := (⌊tl xs⌋, 0), k := ⌈xs ! 0⌉])
    using shift-contents-0 assms(4) by simp
  finally have execute (tm-halve j) (0, tps) (length xs) = (0, tps[j := (⌊tl xs⌋, 0), k := ⌈xs ! 0⌉]) .
  then have execute (tm-halve j) (0, tps) (Suc (length xs)) =
      exe (tm-halve j) (0, tps[j := (⌊tl xs⌋, 0), k := ⌈xs ! 0⌉])
      (is - = exe - (0, ?tps))
    by simp
  also have ... = (1, ?tps[j := (⌊tl xs⌋, 0) |+| 1, k := ⌈𝟬⌉])
    using assms(1,2) exe-cmd-halve-0 by simp
  also have ... = (1, tps[j := (⌊tl xs⌋, 0) |+| 1, k := ⌈𝟬⌉])
    using assms(1,2) by (metis (no-types, opaque-lifting) list-update-overwrite list-update-swap)
  also have ... = (1, tps[j := (⌊tl xs⌋, 1), k := ⌈𝟬⌉])
    by simp
  finally show ?thesis
    using assms(7) by simp
qed

lemma execute-tm-halve:
  assumes j < k
    and length tps = Suc k
    and bit-symbols xs
    and tps ! j = (⌊xs⌋, length xs)
    and tps ! k = ⌈▷⌉
    and tps' = tps[j := (⌊tl xs⌋, 1), k := ⌈𝟬⌉]
  shows execute (tm-halve j) (0, tps) (Suc (length xs)) = (1, tps')
  using execute-cmd-halve-0 execute-tm-halve-1 assms by (cases length xs = 0) simp-all

lemma transforms-tm-halve:
  assumes j < k
    and length tps = Suc k
    and bit-symbols xs
    and tps ! j = (⌊xs⌋, length xs)
    and tps ! k = ⌈▷⌉
    and tps' = tps[j := (⌊tl xs⌋, 1), k := ⌈𝟬⌉]
  shows transforms (tm-halve j) tps (Suc (length xs)) tps'
  using execute-tm-halve assms tm-halve-def transforms-def transits-def by auto

lemma transforms-tm-halve2:
  assumes j < k
    and length tps = k
    and bit-symbols xs
    and tps ! j = (⌊xs⌋, length xs)
    and tps' = tps[j := (⌊tl xs⌋, 1)]
  shows transforms (tm-halve j) (tps @ [⌈▷⌉]) (Suc (length xs)) (tps' @ [⌈𝟬⌉])
proof –
  let ?tps = tps @ [⌈▷⌉]
  let ?tps' = tps' @ [⌈𝟬⌉]
  have ?tps ! j = (⌊xs⌋, length xs) ?tps ! k = ⌈▷⌉
```

```
    using assms by (simp-all add: nth-append)
  moreover have ?tps' ! j = (⌊tl xs⌋, 1) ?tps' ! k = ⌈0⌉
    using assms by (simp-all add: nth-append)
  moreover have length ?tps = Suc k
    using assms(2) by simp
  ultimately show ?thesis
    using assms transforms-tm-halve[OF assms(1), where ?tps=?tps and ?tps'=?tps' and ?xs=xs]
    by (metis length-list-update list-update-append1 list-update-length)
qed
```

The next Turing machine removes the memorization tape from *tm-halve*.

```
definition tm-halve' :: tapeidx ⇒ machine where
  tm-halve' j ≡ cartesian (tm-halve j) 4

lemma bounded-write-tm-halve:
  assumes j < k
  shows bounded-write (tm-halve j) k 4
  unfolding bounded-write-def
proof standard+
  fix q :: nat and rs :: symbol list
  assume q: q < length (tm-halve j) and lenrs: length rs = Suc k
  have k < length rs
    using lenrs by simp
  then have cmd-halve j rs [!] k =
    (if k = j then
        if rs ! j = ▷ then (rs ! k, Right)
        else if last rs = ▷ then (□, Left)
        else (tosym (todigit (last rs)), Left)
      else if k = length rs − 1 then (tosym (todigit (rs ! j)), Stay)
      else (rs ! k, Stay))
    using cmd-halve-def by simp
  then have cmd-halve j rs [!] k = (tosym (todigit (rs ! j)), Stay)
    using assms lenrs by simp
  then have cmd-halve j rs [.] k = tosym (todigit (rs ! j))
    by simp
  moreover have (tm-halve j ! q) rs [.] k = cmd-halve j rs [.] k
    using tm-halve-def q by simp
  ultimately show (tm-halve j ! q) rs [.] k < 4
    by simp
qed

lemma immobile-tm-halve:
  assumes j < k
  shows immobile (tm-halve j) k (Suc k)
proof standard+
  fix q :: nat and rs :: symbol list
  assume q: q < length (tm-halve j) and lenrs: length rs = Suc k
  have k < length rs
    using lenrs by simp
  then have cmd-halve j rs [!] k =
    (if k = j then
        if rs ! j = ▷ then (rs ! k, Right)
        else if last rs = ▷ then (□, Left)
        else (tosym (todigit (last rs)), Left)
      else if k = length rs − 1 then (tosym (todigit (rs ! j)), Stay)
      else (rs ! k, Stay))
    using cmd-halve-def by simp
  then have cmd-halve j rs [!] k = (tosym (todigit (rs ! j)), Stay)
    using assms lenrs by simp
  then have cmd-halve j rs [~] k = Stay
    by simp
  moreover have (tm-halve j ! q) rs [~] k = cmd-halve j rs [~] k
    using tm-halve-def q by simp
```

ultimately show (*tm-halve j* ! *q*) *rs* [~] *k* = *Stay*
  by *simp*
qed

lemma *tm-halve′-tm*:
  assumes *G* ≥ *4* **and** *0* < *j* **and** *j* < *k*
  shows *turing-machine k G* (*tm-halve′ j*)
  using *tm-halve′-def tm-halve-tm assms cartesian-tm* **by** *simp*

lemma *transforms-tm-halve′* [*transforms-intros*]:
  assumes *j* > *0* **and** *j* < *k*
    and *length tps* = *k*
    and *bit-symbols xs*
    and *tps* ! *j* = (⌊*xs*⌋, *length xs*)
    and *tps′* = *tps*[*j* := (⌊*tl xs*⌋, *1*)]
  shows *transforms* (*tm-halve′ j*) *tps* (*Suc* (*length xs*)) *tps′*
  unfolding *tm-halve′-def*
proof (*rule cartesian-transforms-onesie*[*OF tm-halve-tm immobile-tm-halve - - bounded-write-tm-halve assms*(*3*),
where *?G*=*4*];
    (*simp add*: *assms*)*?*)
  show *2* ≤ *k* **and** *2* ≤ *k*
    using *assms*(*1*,*2*) **by** *simp-all*
  show *transforms* (*tm-halve j*) (*tps* @ [⌈*Suc 0*⌉]) (*Suc* (*length xs*))
     (*tps*[*j* := (⌊*tl xs*⌋, *Suc 0*)] @ [⌈**0**⌉])
    using *transforms-tm-halve2 assms* **by** *simp*
qed

lemma *num-tl-div-2*: *num* (*tl xs*) = *num xs div 2*
proof (*cases xs* = [])
  case *True*
  then show *?thesis*
    by (*simp add*: *num-def*)
next
  case *False*
  then have ∗: *xs* = *hd xs* # *tl xs*
    by *simp*
  then have *num xs* = *todigit* (*hd xs*) + *2* ∗ *num* (*tl xs*)
    using *num-Cons* **by** *metis*
  then show *?thesis*
    by *simp*
qed

lemma *canrepr-div-2*: *canrepr* (*n div 2*) = *tl* (*canrepr n*)
  using *canreprI canrepr canonical-canrepr num-tl-div-2 canonical-tl*
  by (*metis hd-Cons-tl list.sel*(*2*))

corollary *nlength-times2*: *nlength* (*2* ∗ *n*) ≤ *Suc* (*nlength n*)
  using *canrepr-div-2*[*of 2* ∗ *n*] **by** *simp*

corollary *nlength-times2plus1*: *nlength* (*2* ∗ *n* + *1*) ≤ *Suc* (*nlength n*)
  using *canrepr-div-2*[*of 2* ∗ *n* + *1*] **by** *simp*

The next Turing machine is the one we actually use to divide a number by two. First it moves to the end of the symbol sequence representing the number, then it applies *tm-halve′*.

definition *tm-div2* :: *tapeidx* ⇒ *machine* **where**
  tm-div2 j ≡ *tm-right-until j* {□} ;; *tm-left j* ;; *tm-halve′ j*

lemma *tm-div2-tm*:
  assumes *G* ≥ *4* **and** *0* < *j* **and** *j* < *k*
  shows *turing-machine k G* (*tm-div2 j*)
  unfolding *tm-div2-def* **using** *tm-right-until-tm tm-left-tm tm-halve′-tm assms* **by** *simp*

locale *turing-machine-div2* =

fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-right-until j* {□}
definition *tm2* ≡ *tm1* ;; *tm-left j*
definition *tm3* ≡ *tm2* ;; *tm-halve′ j*

lemma *tm3-eq-tm-div2*: *tm3* = *tm-div2 j*
unfolding *tm3-def tm2-def tm1-def tm-div2-def* **by** *simp*

context
fixes *tps0* :: *tape list* **and** *k n* :: *nat*
assumes *jk*: *0* < *j j* < *k length tps0* = *k*
and *tps0*: *tps0* ! *j* = (⌊*n*⌋$_N$, *1*)
begin

definition *tps1* ≡ *tps0*
[*j* := (⌊*n*⌋$_N$, *Suc* (*nlength n*))]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *Suc* (*nlength n*)
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
proof (*tform tps*: *tps1-def jk tps0 time*: *assms*)
have *rneigh* (⌊*n*⌋$_N$, *Suc 0*) {□} (*nlength n*)
proof (*intro rneighI*)
show *fst* (⌊*n*⌋$_N$, *Suc 0*) (*snd* (⌊*n*⌋$_N$, *Suc 0*) + *nlength n*) ∈ {□}
using *contents-def* **by** *simp*
show ⋀*n′*. *n′* < *nlength n* ⟹ *fst* (⌊*n*⌋$_N$, *Suc 0*) (*snd* (⌊*n*⌋$_N$, *Suc 0*) + *n′*) ∉ {□}
using *bit-symbols-canrepr contents-def contents-outofbounds proper-symbols-canrepr*
by (*metis One-nat-def Suc-leI add-diff-cancel-left′ fst-eqD less-Suc-eq-0-disj less-nat-zero-code plus-1-eq-Suc singletonD snd-conv*)
qed
then show *rneigh* (*tps0* ! *j*) {□} (*nlength n*)
using *tps0* **by** *simp*
qed

definition *tps2* ≡ *tps0*
[*j* := (⌊*n*⌋$_N$, *nlength n*)]

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *2* + *nlength n*
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def* **by** (*tform tps*: *tps1-def tps2-def jk assms*)

definition *tps3* ≡ *tps0*
[*j* := (⌊*n div 2*⌋$_N$, *1*)]

lemma *tm3*:
assumes *ttt* = *2* ∗ *nlength n* + *3*
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def*
proof (*tform tps*: *tps3-def tps2-def tps0 jk time*: *assms*)
show *bit-symbols* (*canrepr n*)
using *bit-symbols-canrepr* **.**
show *tps3* = *tps2*[*j* := (⌊*tl* (*canrepr n*)⌋, *1*)]
using *tps3-def tps2-def jk tps0 canrepr-div-2* **by** *simp*
qed

end

end

lemma *transforms-tm-div2I* [*transforms-intros*]:
  fixes *tps tps'* :: *tape list* **and** *ttt k n* :: *nat* **and** *j* :: *tapeidx*
  assumes $0 < j$ $j < k$
    and *length tps* = *k*
    and *tps* ! *j* = $(\lfloor n \rfloor_N, 1)$
  assumes *ttt* = 2 ∗ *nlength n* + 3
  assumes *tps'* = *tps*[*j* := $(\lfloor n\ div\ 2 \rfloor_N, 1)$]
  shows *transforms* (*tm-div2 j*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-div2 j* **.**
  show *?thesis*
    using *loc.tm3-eq-tm-div2 loc.tm3 loc.tps3-def assms* **by** *simp*
qed

2.7.10 Modulo two

In this section we construct a Turing machine that writes to tape j_2 the symbol **1** or □ depending on whether the number on tape j_1 is odd or even. If initially tape j_2 contained at most one symbol, it will contain the numbers 1 or 0.

lemma *canrepr-odd*: *odd n* ⟹ *canrepr n* ! *0* = **1**
proof −
  assume *odd n*
  then have $0 < n$
    by *presburger*
  then have *len*: *length* (*canrepr n*) > *0*
    using *nlength-0* **by** *simp*
  then have *canrepr n* ! *0* = **0** ∨ *canrepr n* ! *0* = **1**
    using *bit-symbols-canrepr* **by** *fastforce*
  then show *canrepr n* ! *0* = **1**
    using *prepend-2-even len canrepr* ‹*odd n*› ‹*0* < *n*›
    by (*metis gr0-implies-Suc length-Suc-conv nth-Cons-0*)
qed

lemma *canrepr-even*: *even n* ⟹ *0* < *n* ⟹ *canrepr n* ! *0* = **0**
proof −
  assume *even n* *0* < *n*
  then have *len*: *length* (*canrepr n*) > *0*
    using *nlength-0* **by** *simp*
  then have *canrepr n* ! *0* = **0** ∨ *canrepr n* ! *0* = **1**
    using *bit-symbols-canrepr* **by** *fastforce*
  then show *canrepr n* ! *0* = **0**
    using *prepend-3-odd len canrepr* ‹*even n*› ‹*0* < *n*›
    by (*metis gr0-implies-Suc length-Suc-conv nth-Cons-0*)
qed

definition *tm-mod2 j1 j2* ≡ *tm-trans2 j1 j2* (λ*z*. *if z* = **1** *then* **1** *else* □)

lemma *tm-mod2-tm*:
  assumes $k \geq 2$ **and** $G \geq 4$ **and** *0* < *j2* **and** *j1* < *k* **and** *j2* < *k*
  shows *turing-machine k G* (*tm-mod2 j1 j2*)
  unfolding *tm-mod2-def* **using** *assms tm-trans2-tm* **by** *simp*

lemma *transforms-tm-mod2I* [*transforms-intros*]:
  assumes *j1* < *length tps* **and** *0* < *j2* **and** *j2* < *length tps*
    and $b \leq 1$
  assumes *tps* ! *j1* = $(\lfloor n \rfloor_N, 1)$
    and *tps* ! *j2* = $(\lfloor b \rfloor_N, 1)$
  assumes *tps'* = *tps*[*j2* := $(\lfloor n\ mod\ 2 \rfloor_N, 1)$]
  shows *transforms* (*tm-mod2 j1 j2*) *tps 1 tps'*
proof −
  let *?f* = λ*z*::*symbol*. *if z* = **1** *then* **1** *else* □
  let *?tps* = *tps*[*j2* := *tps* ! *j2* |:=| (*?f* (*tps* :.: *j1*))]
  have ∗: *transforms* (*tm-mod2 j1 j2*) *tps 1 ?tps*

  using *transforms-tm-trans2I assms tm-mod2-def* **by** *metis*

have *tps* :.: *j1* = **1** **if** *odd n*
  using *that canrepr-odd assms*(*5*) *contents-def*
  by (*metis One-nat-def diff-Suc-1 fst-conv gr-implies-not0 ncontents-1-blank-iff-zero odd-pos snd-conv*)
moreover have *tps* :.: *j1* = **0** **if** *even n* **and** *n* > *0*
  using *that canrepr-even assms*(*5*) *contents-def*
  by (*metis One-nat-def diff-Suc-1 fst-conv gr-implies-not0 ncontents-1-blank-iff-zero snd-conv*)
moreover have *tps* :.: *j1* = □ **if** *n* = *0*
  using *that canrepr-even assms*(*5*) *contents-def*
  by *simp*
ultimately have *tps* :.: *j1* = **1** ⟷ *odd n*
  by *linarith*
then have *f*: *?f* (*tps* :.: *j1*) = **1** ⟷ *odd n*
  by *simp*

have *tps-j2*: *tps* ! *j2* |:=| (*?f* (*tps* :.: *j1*)) = ((⌊*b*⌋$_N$)(*1* := (*?f* (*tps* :.: *j1*))), *1*)
  using *assms* **by** *simp*

have *tps* ! *j2* |:=| (*?f* (*tps* :.: *j1*)) = (⌊*n mod 2*⌋$_N$, *1*)
proof (*cases even n*)
  case *True*
  then have *tps* ! *j2* |:=| (*?f* (*tps* :.: *j1*)) = ((⌊*b*⌋$_N$)(*1* := *0*), *1*)
    using *f tps-j2* **by** *auto*
  also have ... = (⌊[]⌋, *1*)
  proof (*cases b* = *0*)
    case *True*
    then have ⌊*b*⌋$_N$ = ⌊[]⌋
      using *canrepr-0* **by** *simp*
    then show *?thesis*
      by *auto*
  next
    case *False*
    then have ⌊*b*⌋$_N$ = ⌊[**1**]⌋
      using *canrepr-1 assms*(*4*) **by** (*metis One-nat-def bot-nat-0.extremum-uniqueI le-Suc-eq*)
    then show *?thesis*
      by (*metis One-nat-def append.simps*(*1*) *append-Nil2 contents-append-update contents-blank-0 list.size*(*3*))
  qed
  also have ... = (⌊*0*⌋$_N$, *1*)
    using *canrepr-0* **by** *simp*
  finally show *?thesis*
    using *True* **by** *auto*
next
  case *False*
  then have *tps* ! *j2* |:=| (*?f* (*tps* :.: *j1*)) = ((⌊*b*⌋$_N$)(*1* := **1**), *1*)
    using *f tps-j2* **by** *auto*
  also have ... = (⌊[**1**]⌋, *1*)
  proof (*cases b* = *0*)
    case *True*
    then have ⌊*b*⌋$_N$ = ⌊[]⌋
      using *canrepr-0* **by** *simp*
    then show *?thesis*
      by (*metis One-nat-def append.simps*(*1*) *contents-snoc list.size*(*3*))
  next
    case *False*
    then have ⌊*b*⌋$_N$ = ⌊[**1**]⌋
      using *canrepr-1 assms*(*4*) **by** (*metis One-nat-def bot-nat-0.extremum-uniqueI le-Suc-eq*)
    then show *?thesis*
      by *auto*
  qed
  also have ... = (⌊*1*⌋$_N$, *1*)
    using *canrepr-1* **by** *simp*
  also have ... = (⌊*n mod 2*⌋$_N$, *1*)

using *False* **by** (*simp add*: *mod2-eq-if*)
finally show *?thesis*
by *auto*
qed
then show *?thesis*
using ∗ *assms*(*7*) **by** *auto*
qed

2.7.11 Boolean operations

In order to support Boolean operations, we represent the value True by the number 1 and False by 0.

abbreviation *bcontents* :: *bool* ⇒ (*nat* ⇒ *symbol*) (‹⌊-⌋$_B$›) **where**
⌊*b*⌋$_B$ ≡ ⌊*if b then 1 else 0*⌋$_N$

A tape containing a number contains the number 0 iff. there is a blank in cell number 1.

lemma *read-ncontents-eq-0*:
assumes *tps* ! *j* = (⌊*n*⌋$_N$, *1*) **and** *j* < *length tps*
shows (*read tps*) ! *j* = □ ⟷ *n* = *0*
using *assms tapes-at-read′*[*of j tps*] *ncontents-1-blank-iff-zero* **by** (*metis prod.sel*(*1*) *prod.sel*(*2*))

And

The next Turing machine, when given two numbers $a, b \in \{0, 1\}$ on tapes j_1 and j_2, writes to tape j_1 the number 1 if $a = b = 1$; otherwise it writes the number 0. In other words, it overwrites tape j_1 with the logical AND of the two tapes.

definition *tm-and* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-and j1 j2 ≡ *IF* λ*rs*. *rs* ! *j1* = **1** ∧ *rs* ! *j2* = □ *THEN tm-write j1* □ *ELSE* [] *ENDIF*

lemma *tm-and-tm*:
assumes *k* ≥ *2* **and** *G* ≥ *4* **and** *0* < *j1* **and** *j1* < *k*
shows *turing-machine k G* (*tm-and j1 j2*)
using *tm-and-def tm-write-tm Nil-tm assms turing-machine-branch-turing-machine* **by** *simp*

locale *turing-machine-and* =
fixes *j1 j2* :: *tapeidx*
begin

context
fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *a b* :: *nat*
assumes *ab*: *a* < *2 b* < *2*
assumes *jk*: *j1* < *k j2* < *k j1* ≠ *j2 0* < *j1 length tps0* = *k*
assumes *tps0*:
tps0 ! *j1* = (⌊*a*⌋$_N$, *1*)
tps0 ! *j2* = (⌊*b*⌋$_N$, *1*)
begin

definition *tps1* ≡ *tps0*
[*j1* := (⌊*a* = *1* ∧ *b* = *1*⌋$_B$, *1*)]

lemma *tm*: *transforms* (*tm-and j1 j2*) *tps0 3 tps1*
unfolding *tm-and-def*
proof (*tform*)
have *read tps0* ! *j1* = ⌊*canrepr a*⌋ *1*
using *jk tps0 tapes-at-read′*[*of j1 tps0*] **by** *simp*
then have *1*: *read tps0* ! *j1* = **1** ⟷ *a* = *1*
using *ab canrepr-odd contents-def ncontents-1-blank-iff-zero*
by (*metis* (*mono-tags*, *lifting*) *One-nat-def diff-Suc-1 less-2-cases-iff odd-one*)
have *read tps0* ! *j2* = ⌊*canrepr b*⌋ *1*
using *jk tps0 tapes-at-read′*[*of j2 tps0*] **by** *simp*
then have *2*: *read tps0* ! *j2* = **1** ⟷ *b* = *1*
using *ab canrepr-odd contents-def ncontents-1-blank-iff-zero*
by (*metis* (*mono-tags*, *lifting*) *One-nat-def diff-Suc-1 less-2-cases-iff odd-one*)

```
  show tps1 = tps0 if ¬ (read tps0 ! j1 = 1 ∧ read tps0 ! j2 = □)
  proof -
    have a = (if a = 1 ∧ b = 1 then 1 else 0)
      using that 1 2 ab jk by (metis One-nat-def less-2-cases-iff read-ncontents-eq-0 tps0(2))
    then have tps0 ! j1 = (⌊a = 1 ∧ b = 1⌋B, 1)
      using tps0 by simp
    then show ?thesis
      unfolding tps1-def using list-update-id[of tps0 j1] by simp
  qed
  show tps1 = tps0[j1 := tps0 ! j1 |:=| □] if read tps0 ! j1 = 1 ∧ read tps0 ! j2 = □
  proof -
    have (if a = 1 ∧ b = 1 then 1 else 0) = 0
      using that 1 2 by simp
    moreover have tps0 ! j1 |:=| □ = (⌊0⌋N, 1)
    proof (cases a = 0)
      case True
      then show ?thesis
        using tps0 jk by auto
    next
      case False
      then have a = 1
        using ab by simp
      then have ⌊a⌋N = ⌊[1]⌋
        using canrepr-1 by simp
      moreover have (⌊[1]⌋, 1) |:=| □ = (⌊[]⌋, 1)
        using contents-def by auto
      ultimately have (⌊a⌋N, 1) |:=| □ = (⌊0⌋N, 1)
        using ncontents-0 by presburger
      then show ?thesis
        using tps0 jk by simp
    qed
    ultimately have tps0 ! j1 |:=| □ = (⌊a = 1 ∧ b = 1⌋B, 1)
      by (smt (verit, best))
    then show ?thesis
      unfolding tps1-def by auto
  qed
qed

end

end

lemma transforms-tm-andI [transforms-intros]:
  fixes j1 j2 :: tapeidx
  fixes tps :: tape list and k :: nat and a b :: nat
  assumes a < 2 b < 2
  assumes length tps = k
  assumes j1 < k j2 < k j1 ≠ j2 0 < j1
  assumes
    tps ! j1 = (⌊a⌋N, 1)
    tps ! j2 = (⌊b⌋N, 1)
  assumes tps' = tps
    [j1 := (⌊a = 1 ∧ b = 1⌋B, 1)]
  shows transforms (tm-and j1 j2) tps 3 tps'
proof -
  interpret loc: turing-machine-and j1 j2 .
  show ?thesis
    using assms loc.tps1-def loc.tm by simp
qed
```

Not

The next Turing machine turns the number 1 into 0 and vice versa.

definition *tm-not* :: *tapeidx* ⇒ *machine* **where**
  tm-not j ≡ *IF* *λrs. rs* ! *j* = □ *THEN tm-write j* **1** *ELSE tm-write j* □ *ENDIF*

lemma *tm-not-tm*:
  assumes $k \geq 2$ **and** $G \geq 4$ **and** $0 < j$ **and** $j < k$
  shows *turing-machine k G* (*tm-not j*)
  using *tm-not-def tm-write-tm assms turing-machine-branch-turing-machine* **by** *simp*

locale *turing-machine-not* =
  fixes *j* :: *tapeidx*
begin

context
  fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *a* :: *nat*
  assumes *a*: $a < 2$
  assumes *jk*: $j < k$ *length tps0* = *k*
  assumes *tps0*: *tps0* ! *j* = ($\lfloor a \rfloor_N$, *1*)
begin

definition *tps1* ≡ *tps0*
  [*j* := ($\lfloor a \neq 1 \rfloor_B$, *1*)]

lemma *tm*: *transforms* (*tm-not j*) *tps0 3 tps1*
  unfolding *tm-not-def*
proof (*tform*)
  have ∗: *read tps0* ! *j* = □ ⟷ *a* = *0*
    using *read-ncontents-eq-0 jk tps0* **by** *simp*
  show *tps1* = *tps0*[*j* := *tps0* ! *j* |:=| **1**] **if** *read tps0* ! *j* = □
  proof −
    have *a* = *0*
      using *a that* ∗ **by** *simp*
    then have ($\lfloor$*if a* = *1 then 0 else 1*$\rfloor_N$, *1*) = ($\lfloor 1 \rfloor_N$, *1*)
      by *simp*
    moreover have *tps0* ! *j* |:=| **1** = ($\lfloor 1 \rfloor_N$, *1*)
      using *tps0 canrepr-0 canrepr-1* ‹*a* = *0*› *contents-snoc*
      by (*metis One-nat-def append.simps*(*1*) *fst-conv list.size*(*3*) *snd-conv*)
    ultimately have *tps0*[*j* := *tps0* ! *j* |:=| **1**] = *tps0*[*j* := ($\lfloor a \neq 1 \rfloor_B$, *1*)]
      by *auto*
    then show *?thesis*
      using *tps1-def* **by** *simp*
  qed
  show *tps1* = *tps0*[*j* := *tps0* ! *j* |:=| □] **if** *read tps0* ! *j* ≠ □
  proof −
    have *a* = *1*
      using *a that* ∗ **by** *simp*
    then have ($\lfloor$*if a* = *1 then 0 else 1*$\rfloor_N$, *1*) = ($\lfloor 0 \rfloor_N$, *1*)
      by *simp*
    moreover have *tps0* ! *j* |:=| □ = ($\lfloor 0 \rfloor_N$, *1*)
      using *tps0 canrepr-0 canrepr-1* ‹*a* = *1*› *contents-snoc*
        by (*metis Suc-1 append-self-conv2 contents-blank-0 fst-eqD fun-upd-upd nat.inject nlength-0-simp numeral-2-eq-2 snd-eqD*)
    ultimately have *tps0*[*j* := *tps0* ! *j* |:=| □] = *tps0*[*j* := ($\lfloor a \neq 1 \rfloor_B$, *1*)]
      by *auto*
    then show *?thesis*
      using *tps1-def* **by** *simp*
  qed
qed

end

end

lemma *transforms-tm-notI* [*transforms-intros*]:
  fixes *j* :: *tapeidx*
  fixes *tps tps'* :: *tape list* **and** *k* :: *nat* **and** *a* :: *nat*
  assumes $j < k$ *length tps* = *k*
    and $a < 2$
  assumes *tps* ! *j* = $(\lfloor a \rfloor_N,\ 1)$
  assumes *tps'* = *tps*
    $[j := (\lfloor a \neq 1 \rfloor_B,\ 1)]$
  shows *transforms* (*tm-not j*) *tps 3 tps'*
proof −
  interpret *loc*: *turing-machine-not j* **.**
  show *?thesis*
    using *assms loc.tps1-def loc.tm* **by** *simp*
qed

end

2.8 Lists of numbers

theory *Lists-Lists*
  imports *Arithmetic*
begin

In the previous section we defined a representation for numbers over the binary alphabet {**0**,**1**}. In this section we first introduce a representation of lists of numbers as symbol sequences over the alphabet {**0**,**1**,|}. Then we define Turing machines for some operations over such lists.

As with the arithmetic operations, Turing machines implementing the operations on lists usually expect the tape heads of the operands to be in position 1 and guarantee to place the tape heads of the result in position 1. The exception are Turing machines for iterating over lists; such TMs move the tape head to the next list element.

A tape containing such representations corresponds to a variable of type *nat list*. A tape in the start configuration corresponds to the empty list of numbers.

2.8.1 Representation as symbol sequence

The obvious idea for representing a list of numbers is to write them one after another separated by a fresh symbol, such as |. However since we represent the number 0 by the empty symbol sequence, this would result in both the empty list and the list containing only the number 0 to be represented by the same symbol sequence, namely the empty one. To prevent this we use the symbol | not as a separator but as a terminator; that is, we append it to every number. Consequently the empty symbol sequence represents the empty list, whereas the list [0] is represented by the symbol sequence |. As another example, the list $[14, 0, 0, 7]$ is represented by **0111**||**111**|. As a side effect of using terminators instead of separators, the representation of the concatenation of lists is just the concatenation of the representations of the individual lists. Moreover the length of the representation is simply the sum of the individual representation lengths. The number of | symbols equals the number of elements in the list.

This is how lists of numbers are represented as symbol sequences:

definition *numlist* :: *nat list* ⇒ *symbol list* **where**
  numlist ns ≡ *concat* (*map* (*λn. canrepr n* @ [|]) *ns*)

lemma *numlist-Nil*: *numlist* [] = []
  using *numlist-def* **by** *simp*

proposition *numlist* [*0*] = [|]
  using *numlist-def* **by** (*simp add*: *canrepr-0*)

lemma *numlist-234*: *set* (*numlist ns*) ⊆ {**0**, **1**, |}
proof (*induction ns*)

case *Nil*
then show *?case*
using *numlist-def* **by** *simp*
next
case (*Cons n ns*)
have *numlist* (*n* # *ns*) = *canrepr n* @ [|] @ *concat* (*map* (λ*n*. *canrepr n* @ [|]) *ns*)
using *numlist-def* **by** *simp*
then have *numlist* (*n* # *ns*) = *canrepr n* @ [|] @ *numlist ns*
using *numlist-def* **by** *simp*
moreover have *set* ([|] @ *numlist ns*) $\subseteq$ {**0**, **1**, |}
using *Cons* **by** *simp*
moreover have *set* (*canrepr n*) $\subseteq$ {**0**, **1**, |}
using *bit-symbols-canrepr* **by** (*metis in-set-conv-nth insertCI subsetI*)
ultimately show *?case*
by *simp*
qed

lemma *symbols-lt-numlist*: *symbols-lt 5* (*numlist ns*)
using *numlist-234*
by (*metis empty-iff insert-iff nth-mem numeral-less-iff semiring-norm*(*68*) *semiring-norm*(*76*) *semiring-norm*(*79*) *semiring-norm*(*80*) *subset-code*(*1*) *verit-comp-simplify1*(*2*))

lemma *bit-symbols-prefix-eq*:
assumes (*x* @ [|]) @ *xs* = (*y* @ [|]) @ *ys* **and** *bit-symbols x* **and** *bit-symbols y*
shows *x* = *y*
proof −
have ∗: *x* @ [|] @ *xs* = *y* @ [|] @ *ys*
(**is** *?lhs* = *?rhs*)
using *assms*(*1*) **by** *simp*
show *x* = *y*
proof (*cases length x* ≤ *length y*)
case *True*
then have *?lhs* ! *i* = *?rhs* ! *i* **if** *i* < *length x* **for** *i*
using *that* ∗ **by** *simp*
then have *eq*: *x* ! *i* = *y* ! *i* **if** *i* < *length x* **for** *i*
using *that True* **by** (*metis Suc-le-eq le-trans nth-append*)
have *?lhs* ! (*length x*) = |
by (*metis Cons-eq-appendI nth-append-length*)
then have *?rhs* ! (*length x*) = |
using ∗ **by** *metis*
then have *length y* ≤ *length x*
using *assms*(*3*)
by (*metis linorder-le-less-linear nth-append numeral-eq-iff semiring-norm*(*85*) *semiring-norm*(*87*) *semiring-norm*(*89*))
then have *length y* = *length x*
using *True* **by** *simp*
then show *?thesis*
using *eq* **by** (*simp add*: *list-eq-iff-nth-eq*)
next
case *False*
then have *?lhs* ! *i* = *?rhs* ! *i* **if** *i* < *length y* **for** *i*
using *that* ∗ **by** *simp*
have *?rhs* ! (*length y*) = |
by (*metis Cons-eq-appendI nth-append-length*)
then have *?lhs* ! (*length y*) = |
using ∗ **by** *metis*
then have *x* ! (*length y*) = |
using *False* **by** (*simp add*: *nth-append*)
then have *False*
using *assms*(*2*) *False*
by (*metis linorder-le-less-linear numeral-eq-iff semiring-norm*(*85*) *semiring-norm*(*87*) *semiring-norm*(*89*))
then show *?thesis*
by *simp*

qed
qed

lemma *numlist-inj*: *numlist ns1* = *numlist ns2* $\Longrightarrow$ *ns1* = *ns2*
proof (*induction ns1 arbitrary*: *ns2*)
case *Nil*
then show *?case*
using *numlist-def* **by** *simp*
next
case (*Cons n ns1*)
have *1*: *numlist* (*n* # *ns1*) = (*canrepr n* @ [|]) @ *numlist ns1*
using *numlist-def* **by** *simp*
then have *numlist ns2* = (*canrepr n* @ [|]) @ *numlist ns1*
using *Cons* **by** *simp*
then have *ns2* $\neq$ []
using *numlist-Nil* **by** *auto*
then have *2*: *ns2* = *hd ns2* # *tl ns2*
using ‹*ns2* $\neq$ []› **by** *simp*
then have *3*: *numlist ns2* = (*canrepr* (*hd ns2*) @ [|]) @ *numlist* (*tl ns2*)
using *numlist-def* **by** (*metis concat.simps*(*2*) *list.simps*(*9*))

have *4*: *hd ns2* = *n*
using *bit-symbols-prefix-eq 1 3* **by** (*metis Cons.prems canrepr bit-symbols-canrepr*)
then have *numlist ns2* = (*canrepr n* @ [|]) @ *numlist* (*tl ns2*)
using *3* **by** *simp*
then have *numlist ns1* = *numlist* (*tl ns2*)
using *1* **by** (*simp add*: *Cons.prems*)
then have *ns1* = *tl ns2*
using *Cons* **by** *simp*
then show *?case*
using *2 4* **by** *simp*
qed

corollary *proper-symbols-numlist*: *proper-symbols* (*numlist ns*)
using *numlist-234 nth-mem* **by** *fastforce*

The next property would not hold if we used separators between elements instead of terminators after elements.

lemma *numlist-append*: *numlist* (*xs* @ *ys*) = *numlist xs* @ *numlist ys*
using *numlist-def* **by** *simp*

Like *nlength* for numbers, we have *nllength* for the length of the representation of a list of numbers.

definition *nllength* :: *nat list* $\Rightarrow$ *nat* **where**
nllength ns $\equiv$ *length* (*numlist ns*)

lemma *nllength*: *nllength ns* = ($\sum$ *n*$\leftarrow$*ns*. *Suc* (*nlength n*))
using *nllength-def numlist-def* **by** (*induction ns*) *simp-all*

lemma *nllength-Nil* [*simp*]: *nllength* [] = *0*
using *nllength-def numlist-def* **by** *simp*

lemma *length-le-nllength*: *length ns* $\leq$ *nllength ns*
using *nllength sum-list-mono*[*of ns* λ-. *1* λ*n*. *Suc* (*nlength n*)] *sum-list-const*[*of 1 ns*]
by *simp*

lemma *nllength-le-len-mult-max*:
fixes *N* :: *nat* **and** *ns* :: *nat list*
assumes $\forall$ *n*$\in$*set ns*. *n* $\leq$ *N*
shows *nllength ns* $\leq$ *Suc* (*nlength N*) $*$ *length ns*
proof −
have *nllength ns* = ($\sum$ *n*$\leftarrow$*ns*. *Suc* (*nlength n*))
using *nllength* **by** *simp*
moreover have *Suc* (*nlength n*) $\leq$ *Suc* (*nlength N*) **if** *n* $\in$ *set ns* **for** *n*

using *nlength-mono that assms* **by** *simp*
ultimately have *nllength ns* $\leq$ ($\sum$ *n*$\leftarrow$*ns. Suc* (*nlength N*))
by (*simp add*: *sum-list-mono*)
then show *nllength ns* $\leq$ *Suc* (*nlength N*) $*$ *length ns*
using *sum-list-const* **by** *metis*
qed

lemma *nllength-upt-le-len-mult-max*:
fixes *a b* :: *nat*
shows *nllength* [*a..<b*] $\leq$ *Suc* (*nlength b*) $*$ (*b* $-$ *a*)
using *nllength-le-len-mult-max*[*of* [*a..<b*] *b*] **by** *simp*

lemma *nllength-le-len-mult-Max*: *nllength ns* $\leq$ *Suc* (*nlength* (*Max* (*set ns*))) $*$ *length ns*
using *nllength-le-len-mult-max* **by** *simp*

lemma *member-le-nllength*: *n* $\in$ *set ns* $\Longrightarrow$ *nlength n* $\leq$ *nllength ns*
using *nllength* **by** (*induction ns*) *auto*

lemma *member-le-nllength-1*: *n* $\in$ *set ns* $\Longrightarrow$ *nlength n* $\leq$ *nllength ns* $-$ *1*
using *nllength* **by** (*induction ns*) *auto*

lemma *nllength-gr-0*: *ns* $\neq$ [] $\Longrightarrow$ *0* $<$ *nllength ns*
using *nllength-def numlist-def* **by** *simp*

lemma *nlength-min-le-nllength*: *n* $\in$ *set ns* $\Longrightarrow$ *m* $\in$ *set ns* $\Longrightarrow$ *nlength* (*min n m*) $\leq$ *nllength ns*
using *member-le-nllength* **by** (*simp add*: *min-def*)

lemma *take-drop-numlist*:
assumes *i* $<$ *length ns*
shows *take* (*Suc* (*nlength* (*ns* ! *i*))) (*drop* (*nllength* (*take i ns*)) (*numlist ns*)) = *canrepr* (*ns* ! *i*) @ [|]
proof $-$
have *numlist ns* = *numlist* (*take i ns*) @ *numlist* (*drop i ns*)
using *numlist-append* **by** (*metis append-take-drop-id*)
moreover have *numlist* (*drop i ns*) = *numlist* [*ns* ! *i*] @ *numlist* (*drop* (*Suc i*) *ns*)
using *assms numlist-append* **by** (*metis Cons-nth-drop-Suc append-Cons self-append-conv2*)
ultimately have *numlist ns* = *numlist* (*take i ns*) @ *numlist* [*ns* ! *i*] @ *numlist* (*drop* (*Suc i*) *ns*)
by *simp*
then have *drop* (*nllength* (*take i ns*)) (*numlist ns*) = *numlist* [*ns* ! *i*] @ *numlist* (*drop* (*Suc i*) *ns*)
by (*simp add*: *nllength-def*)
then have *drop* (*nllength* (*take i ns*)) (*numlist ns*) = *canrepr* (*ns* ! *i*) @ [|] @ *numlist* (*drop* (*Suc i*) *ns*)
using *numlist-def* **by** *simp*
then show *?thesis*
by *simp*
qed

corollary *take-drop-numlist′*:
assumes *i* $<$ *length ns*
shows *take* (*nlength* (*ns* ! *i*)) (*drop* (*nllength* (*take i ns*)) (*numlist ns*)) = *canrepr* (*ns* ! *i*)
using *take-drop-numlist*[*OF assms*] **by** (*metis append-assoc append-eq-conv-conj append-take-drop-id*)

corollary *numlist-take-at-term*:
assumes *i* $<$ *length ns*
shows *numlist ns* ! (*nllength* (*take i ns*) + *nlength* (*ns* ! *i*)) = |
using *assms take-drop-numlist nllength-def numlist-append*
by (*smt* (*verit*) *append-eq-conv-conj append-take-drop-id lessI nth-append-length nth-append-length-plus nth-take*)

lemma *numlist-take-at*:
assumes *i* $<$ *length ns* **and** *j* $<$ *nlength* (*ns* ! *i*)
shows *numlist ns* ! (*nllength* (*take i ns*) + *j*) = *canrepr* (*ns* ! *i*) ! *j*
proof $-$
have *ns* = *take i ns* @ [*ns* ! *i*] @ *drop* (*Suc i*) *ns*
using *assms* **by** (*metis Cons-eq-appendI append-self-conv2 id-take-nth-drop*)
then have *numlist ns* = (*numlist* (*take i ns*) @ *numlist* [*ns* ! *i*]) @ *numlist* (*drop* (*Suc i*) *ns*)

using *numlist-append* **by** (*metis append-assoc*)
moreover have *nllength* (*take i ns*) + *j* < *nllength* (*take i ns*) + *nllength* [*ns* ! *i*]
using *assms*(*2*) *nllength* **by** *simp*
ultimately have *numlist ns* ! (*nllength* (*take i ns*) + *j*) =
(*numlist* (*take i ns*) @ *numlist* [*ns* ! *i*]) ! (*nllength* (*take i ns*) + *j*)
by (*metis length-append nllength-def nth-append*)
also have ... = *numlist* [*ns* ! *i*] ! *j*
by (*simp add*: *nllength-def*)
also have ... = (*canrepr* (*ns* ! *i*) @ [|]) ! *j*
using *numlist-def* **by** *simp*
also have ... = *canrepr* (*ns* ! *i*) ! *j*
using *assms*(*2*) **by** (*simp add*: *nth-append*)
finally show *?thesis* **.**
qed

lemma *nllength-take-Suc*:
assumes *i* < *length ns*
shows *nllength* (*take i ns*) + *Suc* (*nlength* (*ns* ! *i*)) = *nllength* (*take* (*Suc i*) *ns*)
proof −
have *ns* = *take i ns* @ [*ns* ! *i*] @ *drop* (*Suc i*) *ns*
using *assms* **by** (*metis Cons-eq-appendI append-self-conv2 id-take-nth-drop*)
then have *numlist ns* = *numlist* (*take i ns*) @ *numlist* [*ns* ! *i*] @ *numlist* (*drop* (*Suc i*) *ns*)
using *numlist-append* **by** *metis*
then have *nllength ns* = *nllength* (*take i ns*) + *nllength* [*ns* ! *i*] + *nllength* (*drop* (*Suc i*) *ns*)
by (*simp add*: *nllength-def*)
moreover have *nllength ns* = *nllength* (*take* (*Suc i*) *ns*) + *nllength* (*drop* (*Suc i*) *ns*)
using *numlist-append* **by** (*metis append-take-drop-id length-append nllength-def*)
ultimately have *nllength* (*take* (*Suc i*) *ns*) = *nllength* (*take i ns*) + *nllength* [*ns* ! *i*]
by *simp*
then show *?thesis*
using *nllength* **by** *simp*
qed

lemma *numlist-take-Suc-at-term*:
assumes *i* < *length ns*
shows *numlist ns* ! (*nllength* (*take* (*Suc i*) *ns*) − *1*) = |
proof −
have *nllength* (*take* (*Suc i*) *ns*) − *1* = *nllength* (*take i ns*) + *nlength* (*ns* ! *i*)
using *nllength-take-Suc assms* **by** (*metis add-Suc-right diff-Suc-1*)
then show *?thesis*
using *numlist-take-at-term assms* **by** *simp*
qed

lemma *nllength-take*:
assumes *i* < *length ns*
shows *nllength* (*take i ns*) + *nlength* (*ns* ! *i*) < *nllength ns*
proof −
have *ns* = *take i ns* @ [*ns* ! *i*] @ *drop* (*Suc i*) *ns*
using *assms* **by** (*metis Cons-eq-appendI append-self-conv2 id-take-nth-drop*)
then have *numlist ns* = *numlist* (*take i ns*) @ *numlist* [*ns* ! *i*] @ *numlist* (*drop* (*Suc i*) *ns*)
using *numlist-append* **by** *metis*
then have *nllength ns* = *nllength* (*take i ns*) + *nllength* [*ns* ! *i*] + *nllength* (*drop* (*Suc i*) *ns*)
by (*simp add*: *nllength-def*)
then have *nllength ns* ≥ *nllength* (*take i ns*) + *nllength* [*ns* ! *i*]
by *simp*
then have *nllength ns* ≥ *nllength* (*take i ns*) + *Suc* (*nlength* (*ns* ! *i*))
using *nllength* **by** *simp*
then show *?thesis*
by *simp*
qed

The contents of a tape starting with the start symbol ▷ followed by the symbol sequence representing a list of numbers:

definition *nlcontents* :: *nat list* ⇒ (*nat* ⇒ *symbol*) (‹⌊-⌋$_{NL}$›) **where**
  ⌊*ns*⌋$_{NL}$ ≡ ⌊*numlist ns*⌋

lemma *clean-tape-nlcontents*: *clean-tape* (⌊*ns*⌋$_{NL}$, *i*)
  by (*simp add*: *nlcontents-def proper-symbols-numlist*)

lemma *nlcontents-Nil*: ⌊[]⌋$_{NL}$ = ⌊[]⌋
  using *nlcontents-def* **by** (*simp add*: *numlist-Nil*)

lemma *nlcontents-rneigh-4*:
  assumes *i* < *length ns*
  shows *rneigh* (⌊*ns*⌋$_{NL}$, *Suc* (*nllength* (*take i ns*))) {|} (*nlength* (*ns* ! *i*))
proof (*rule rneighI*)
  let *?tp* = (⌊*ns*⌋$_{NL}$, *Suc* (*nllength* (*take i ns*)))
  show *fst ?tp* (*snd ?tp* + *nlength* (*ns* ! *i*)) ∈ {|}
  proof −
    have *snd ?tp* + *nlength* (*ns* ! *i*) ≤ *nllength ns*
      using *nllength-take assms* **by** (*simp add*: *Suc-leI*)
    then have *fst ?tp* (*snd ?tp* + *nlength* (*ns* ! *i*)) = *numlist ns* ! (*nllength* (*take i ns*) + *nlength* (*ns* ! *i*))
      using *nlcontents-def contents-inbounds nllength-def* **by** *simp*
    then show *?thesis*
      using *numlist-take-at-term assms* **by** *simp*
  qed
  show *fst ?tp* (*snd ?tp* + *j*) ∉ {|} **if** *j* < *nlength* (*ns* ! *i*) **for** *j*
  proof −
    have *snd ?tp* + *nlength* (*ns* ! *i*) ≤ *nllength ns*
      using *nllength-take assms* **by** (*simp add*: *Suc-leI*)
    then have *snd ?tp* + *j* ≤ *nllength ns*
      using *that* **by** *simp*
    then have *fst ?tp* (*snd ?tp* + *j*) = *numlist ns* ! (*nllength* (*take i ns*) + *j*)
      using *nlcontents-def contents-inbounds nllength-def* **by** *simp*
    then have *fst ?tp* (*snd ?tp* + *j*) = *canrepr* (*ns* ! *i*) ! *j*
      using *assms that numlist-take-at* **by** *simp*
    then show *?thesis*
      using *bit-symbols-canrepr that* **by** *fastforce*
  qed
qed

lemma *nlcontents-rneigh-04*:
  assumes *i* < *length ns*
  shows *rneigh* (⌊*ns*⌋$_{NL}$, *Suc* (*nllength* (*take i ns*))) {□, |} (*nlength* (*ns* ! *i*))
proof (*rule rneighI*)
  let *?tp* = (⌊*ns*⌋$_{NL}$, *Suc* (*nllength* (*take i ns*)))
  show *fst ?tp* (*snd ?tp* + *nlength* (*ns* ! *i*)) ∈ {□, |}
  proof −
    have *snd ?tp* + *nlength* (*ns* ! *i*) ≤ *nllength ns*
      using *nllength-take assms* **by** (*simp add*: *Suc-leI*)
    then have *fst ?tp* (*snd ?tp* + *nlength* (*ns* ! *i*)) = *numlist ns* ! (*nllength* (*take i ns*) + *nlength* (*ns* ! *i*))
      using *nlcontents-def contents-inbounds nllength-def* **by** *simp*
    then show *?thesis*
      using *numlist-take-at-term assms* **by** *simp*
  qed
  show *fst ?tp* (*snd ?tp* + *j*) ∉ {□, |} **if** *j* < *nlength* (*ns* ! *i*) **for** *j*
  proof −
    have *snd ?tp* + *nlength* (*ns* ! *i*) ≤ *nllength ns*
      using *nllength-take assms* **by** (*simp add*: *Suc-leI*)
    then have *snd ?tp* + *j* ≤ *nllength ns*
      using *that* **by** *simp*
    then have *fst ?tp* (*snd ?tp* + *j*) = *numlist ns* ! (*nllength* (*take i ns*) + *j*)
      using *nlcontents-def contents-inbounds nllength-def* **by** *simp*
    then have *fst ?tp* (*snd ?tp* + *j*) = *canrepr* (*ns* ! *i*) ! *j*
      using *assms that numlist-take-at* **by** *simp*
    then show *?thesis*

using *bit-symbols-canrepr that* **by** *fastforce*
qed
qed

A tape storing a list of numbers, with the tape head in the first blank cell after the symbols:

abbreviation *nltape* :: *nat list* ⇒ *tape* **where**
nltape ns ≡ $(\lfloor ns \rfloor_{NL}$, *Suc* (*nllength ns*))

A tape storing a list of numbers, with the tape head on the first symbol representing the i-th number, unless the i-th number is zero, in which case the tape head is on the terminator symbol of this zero. If i is out of bounds of the list, the tape head is at the first blank after the list.

abbreviation *nltape'* :: *nat list* ⇒ *nat* ⇒ *tape* **where**
nltape' ns i ≡ $(\lfloor ns \rfloor_{NL}$, *Suc* (*nllength* (*take i ns*)))

lemma *nltape'-tape-read*: |.| (*nltape' ns i*) = □ ⟷ *i* ≥ *length ns*
proof −
have |.| (*nltape' ns i*) = □ **if** *i* ≥ *length ns* **for** *i*
proof −
have *nltape' ns i* ≡ $(\lfloor ns \rfloor_{NL}$, *Suc* (*nllength ns*))
using *that* **by** *simp*
then show *?thesis*
using *nlcontents-def contents-outofbounds nllength-def*
by (*metis Suc-eq-plus1 add.left-neutral fst-conv less-Suc0 less-add-eq-less snd-conv*)
qed
moreover have |.| (*nltape' ns i*) ≠ □ **if** *i* < *length ns* **for** *i*
using *that nlcontents-def contents-inbounds nllength-def nllength-take proper-symbols-numlist*
by (*metis Suc-leI add-Suc-right diff-Suc-1 fst-conv less-add-same-cancel1 less-le-trans not-add-less2 plus-1-eq-Suc snd-conv zero-less-Suc*)
ultimately show *?thesis*
by (*meson le-less-linear*)
qed

2.8.2 Moving to the next element

The next Turing machine provides a means to iterate over a list of numbers. If the TM starts in a configuration where the tape j_1 contains a list of numbers and the tape head is on the first symbol of the i-th element of this list, then after the TM has finished, the i-th element will be written on tape j_2 and the tape head on j_1 will have advanced by one list element. If i is the last element of the list, the tape head on j_1 will be on a blank symbol. One can execute this TM in a loop until the tape head reaches a blank. The TM is generic over a parameter z representing the terminator symbol, so it can be used for other kinds of lists, too (see Section 2.9).

definition *tm-nextract* :: *symbol* ⇒ *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-nextract z j1 j2 ≡
tm-erase-cr j2 ;;
tm-cp-until j1 j2 {*z*} ;;
tm-cr j2 ;;
tm-right j1

lemma *tm-nextract-tm*:
assumes *G* ≥ *4* **and** *G* > *z* **and** *0* < *j2* **and** *j2* < *k* **and** *j1* < *k* **and** *k* ≥ *2*
shows *turing-machine k G* (*tm-nextract z j1 j2*)
unfolding *tm-nextract-def*
using *assms tm-erase-cr-tm tm-cp-until-tm tm-cr-tm tm-right-tm*
by *simp*

The next locale is for the case $z = |$.

locale *turing-machine-nextract-4* =
fixes *j1 j2* :: *tapeidx*
begin

definition *tm1* ≡ *tm-erase-cr j2*
definition *tm2* ≡ *tm1* ;; *tm-cp-until j1 j2* {|}
definition *tm3* ≡ *tm2* ;; *tm-cr j2*
definition *tm4* ≡ *tm3* ;; *tm-right j1*

lemma *tm4-eq-tm-nextract*: *tm4* = *tm-nextract* | *j1 j2*
  unfolding *tm1-def tm2-def tm3-def tm4-def tm-nextract-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *k idx dummy* :: *nat* **and** *ns* :: *nat list*
  assumes *jk*: *j1* < *k j2* < *k 0* < *j1 0* < *j2 j1* ≠ *j2 length tps0* = *k*
    and *idx*: *idx* < *length ns*
    and *tps0*:
      tps0 ! *j1* = *nltape′ ns idx*
      tps0 ! *j2* = (⌊*dummy*⌋$_N$, *1*)
begin

definition *tps1* ≡ *tps0*[*j2* := (⌊*0*⌋$_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
  assumes *ttt* = *7* + *2* ∗ *nlength dummy*
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def*
proof (*tform tps*: *jk idx tps0 tps1-def assms*)
  show *proper-symbols* (*canrepr dummy*)
    using *proper-symbols-canrepr* **by** *simp*
  show *tps1* = *tps0*[*j2* := (⌊[]⌋, *1*)]
    using *ncontents-0 tps1-def* **by** *simp*
qed

definition *tps2* ≡ *tps0*
  [*j1* := (⌊*ns*⌋$_{NL}$, *nllength* (*take* (*Suc idx*) *ns*)),
   j2 := (⌊*ns* ! *idx*⌋$_N$, *Suc* (*nlength* (*ns* ! *idx*)))]

lemma *tm2* [*transforms-intros*]:
  assumes *ttt* = *7* + *2* ∗ *nlength dummy* + *Suc* (*nlength* (*ns* ! *idx*))
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def*
proof (*tform tps*: *jk idx tps0 tps2-def tps1-def*)
  show *rneigh* (*tps1* ! *j1*) {|} (*nlength* (*ns* ! *idx*))
    using *tps1-def tps0 jk* **by** (*simp add*: *idx nlcontents-rneigh-4*)
  show *tps2* = *tps1*
    [*j1* := *tps1* ! *j1* |+| *nlength* (*ns* ! *idx*),
     j2 := *implant* (*tps1* ! *j1*) (*tps1* ! *j2*) (*nlength* (*ns* ! *idx*))]
     (**is** *?l* = *?r*)
  proof (*rule nth-equalityI*)
    show *len*: *length ?l* = *length ?r*
      using *tps1-def tps2-def* **by** *simp*
    show *?l* ! *i* = *?r* ! *i* **if** *i* < *length ?l* **for** *i*
    proof −
      consider *i* = *j1* | *i* = *j2* | *i* ≠ *j1* ∧ *i* ≠ *j2*
        by *auto*
      then show *?thesis*
      proof (*cases*)
        case *1*
        then show *?thesis*
          using *tps0 that tps1-def tps2-def jk nllength-take-Suc*[*OF idx*] *idx* **by** *simp*
      next
        case *2*
        then have *lhs*: *?l* ! *i* = (⌊*ns* ! *idx*⌋$_N$, *Suc* (*nlength* (*ns* ! *idx*)))
          using *tps2-def jk* **by** *simp*
        let *?i* = *Suc* (*nllength* (*take idx ns*))

```
        have i1: ?i > 0
          by simp
        have nlength (ns ! idx) + (?i − 1) ≤ nllength ns
          using idx by (simp add: add.commute less-or-eq-imp-le nllength-take)
        then have i2: nlength (ns ! idx) + (?i − 1) ≤ length (numlist ns)
          using nllength-def by simp
        have ?r ! i = implant (tps1 ! j1) (tps1 ! j2) (nlength (ns ! idx))
          using 2 tps1-def jk by simp
        also have ... = implant (⌊ns⌋NL, ?i) (⌊0⌋N, 1) (nlength (ns ! idx))
          using tps1-def jk tps0 by simp
        also have ... =
          (⌊[] @ (take (nlength (ns ! idx)) (drop (?i − 1) (numlist ns)))⌋,
           Suc (length []) + nlength (ns ! idx))
          using implant-contents[OF i1 i2] by (metis One-nat-def list.size(3) ncontents-0 nlcontents-def)
        finally have ?r ! i =
          (⌊[] @ (take (nlength (ns ! idx)) (drop (?i − 1) (numlist ns)))⌋,
           Suc (length []) + nlength (ns ! idx)) .
        then have ?r ! i = (⌊take (nlength (ns ! idx)) (drop (nllength (take idx ns)) (numlist ns))⌋, Suc (nlength
(ns ! idx)))
          by simp
        then have ?r ! i = (⌊canrepr (ns ! idx)⌋, Suc (nlength (ns ! idx)))
          using take-drop-numlist'[OF idx] by simp
        then show ?thesis
          using lhs by simp
      next
        case 3
        then show ?thesis
          using that tps1-def tps2-def jk by simp
      qed
    qed
  qed
  show ttt = 7 + 2 ∗ nlength dummy + Suc (nlength (ns ! idx))
    using assms(1) .
qed

definition tps3 ≡ tps0
  [j1 := (⌊ns⌋NL, nllength (take (Suc idx) ns)),
   j2 := (⌊ns ! idx⌋N, 1)]

lemma tm3 [transforms-intros]:
  assumes ttt = 11 + 2 ∗ nlength dummy + 2 ∗ (nlength (ns ! idx))
  shows transforms tm3 tps0 ttt tps3
  unfolding tm3-def
proof (tform tps: jk idx tps0 tps2-def tps3-def time: assms tps2-def jk)
  show clean-tape (tps2 ! j2)
    using tps2-def jk clean-tape-ncontents by simp
qed

definition tps4 ≡ tps0
  [j1 := nltape' ns (Suc idx),
   j2 := (⌊ns ! idx⌋N, 1)]

lemma tm4:
  assumes ttt = 12 + 2 ∗ nlength dummy + 2 ∗ (nlength (ns ! idx))
  shows transforms tm4 tps0 ttt tps4
  unfolding tm4-def
proof (tform tps: jk idx tps0 tps3-def tps4-def time: assms)
  show tps4 = tps3[j1 := tps3 ! j1 |+| 1]
    using tps3-def tps4-def jk tps0
      by (metis Suc-eq-plus1 fst-conv list-update-overwrite list-update-swap nth-list-update-eq nth-list-update-neq
snd-conv)
qed
```

end

end

lemma *transforms-tm-nextract-4I* [*transforms-intros*]:
  fixes *j1 j2* :: *tapeidx*
  fixes *tps tps'* :: *tape list* **and** *k idx dummy* :: *nat* **and** *ns* :: *nat list*
  assumes $j1 < k$ $j2 < k$ $0 < j1$ $0 < j2$ $j1 \neq j2$ *length tps* $= k$
    and *idx* < *length ns*
  assumes
    tps ! *j1* = *nltape' ns idx*
    tps ! *j2* = (⌊*dummy*⌋$_N$, *1*)
  assumes *ttt* = *12* + *2* ∗ *nlength dummy* + *2* ∗ (*nlength* (*ns* ! *idx*))
  assumes *tps'* = *tps*
    [*j1* := *nltape' ns* (*Suc idx*),
     j2 := (⌊*ns* ! *idx*⌋$_N$, *1*)]
  shows *transforms* (*tm-nextract* | *j1 j2*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-nextract-4 j1 j2* **.**
  show *?thesis*
    using *assms loc.tm4 loc.tps4-def loc.tm4-eq-tm-nextract* **by** *simp*
qed

2.8.3 Appending an element

The next Turing machine appends the number on tape j_2 to the list of numbers on tape j_1.

definition *tm-append* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-append j1 j2 ≡
    tm-right-until j1 {□} ;;
    tm-cp-until j2 j1 {□} ;;
    tm-cr j2 ;;
    tm-char j1 |

lemma *tm-append-tm*:
  assumes $0 < j1$ **and** $G \geq 5$ **and** $j1 < k$ **and** $j2 < k$
  shows *turing-machine k G* (*tm-append j1 j2*)
  unfolding *tm-append-def*
  using *assms tm-right-until-tm tm-cp-until-tm tm-right-tm tm-char-tm tm-cr-tm*
  by *simp*

locale *turing-machine-append* =
  fixes *j1 j2* :: *tapeidx*
begin

definition *tm1* ≡ *tm-right-until j1* {□}
definition *tm2* ≡ *tm1* ;; *tm-cp-until j2 j1* {□}
definition *tm3* ≡ *tm2* ;; *tm-cr j2*
definition *tm4* ≡ *tm3* ;; *tm-char j1* |

lemma *tm4-eq-tm-append*: *tm4* = *tm-append j1 j2*
  unfolding *tm4-def tm3-def tm2-def tm1-def tm-append-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *k i1 n* :: *nat* **and** *ns* :: *nat list*
  assumes *jk*: *length tps0* = k $j1 < k$ $j2 < k$ $j1 \neq j2$ $0 < j1$
    and *i1*: $i1 \leq$ *Suc* (*nllength ns*)
    and *tps0*:
      tps0 ! *j1* = (⌊*ns*⌋$_{NL}$, *i1*)
      tps0 ! *j2* = (⌊*n*⌋$_N$, *1*)
begin

lemma *k*: $k \geq 2$
  using *jk* **by** *simp*

```
lemma tm1 [transforms-intros]:
  assumes ttt = Suc (Suc (nllength ns) − i1)
    and tps' = tps0[j1 := nltape ns]
  shows transforms tm1 tps0 ttt tps'
  unfolding tm1-def
proof (tform tps: jk k)
  let ?l = Suc (nllength ns) − i1
  show rneigh (tps0 ! j1) {0} ?l
  proof (rule rneighI)
    show (tps0 ::: j1) (tps0 :#: j1 + ?l) ∈ {0}
      using tps0 jk nlcontents-def nllength-def by simp
    show (tps0 ::: j1) (tps0 :#: j1 + i) ∉ {0} if i < Suc (nllength ns) − i1 for i
    proof −
      have i1 + i < Suc (nllength ns)
        using that i1 by simp
      then show ?thesis
        using proper-symbols-numlist nllength-def tps0 nlcontents-def contents-def
        by (metis One-nat-def Suc-le-eq diff-Suc-1 fst-eqD less-Suc-eq-0-disj less-nat-zero-code singletonD snd-eqD)
    qed
  qed
  show ttt = Suc (Suc (nllength ns) − i1)
    using assms(1) .
  show tps' = tps0[j1 := tps0 ! j1 |+| Suc (nllength ns) − i1]
    using assms(2) tps0 i1 by simp
qed

lemma tm2 [transforms-intros]:
  assumes ttt = 3 + nllength ns − i1 + nlength n
    and tps' = tps0
      [j1 := (⌊numlist ns @ canrepr n⌋, Suc (nllength ns) + nlength n),
       j2 := (⌊n⌋N, Suc (nlength n))]
  shows transforms tm2 tps0 ttt tps'
  unfolding tm2-def
proof (tform tps: jk tps0)
  let ?tps = tps0[j1 := nltape ns]
  have j1: ?tps ! j1 = nltape ns
    by (simp add: jk)
  have j2: ?tps ! j2 = (⌊n⌋N, 1)
    using tps0 jk by simp
  show rneigh (tps0[j1 := nltape ns] ! j2) {0} (nlength n)
  proof (rule rneighI)
    show (?tps ::: j2) (?tps :#: j2 + nlength n) ∈ {0}
      using j2 contents-outofbounds by simp
    show ⋀i. i < nlength n ⟹ (?tps ::: j2) (?tps :#: j2 + i) ∉ {0}
      using j2 tps0 bit-symbols-canrepr by fastforce
  qed
  show tps' = tps0
    [j1 := nltape ns,
     j2 := ?tps ! j2 |+| nlength n,
     j1 := implant (?tps ! j2) (?tps ! j1) (nlength n)]
    (is - = ?rhs)
  proof −
    have implant (?tps ! j2) (?tps ! j1) (nlength n) = implant (⌊n⌋N, 1) (nltape ns) (nlength n)
      using j1 j2 by simp
    also have ... = (⌊numlist ns @ (take (nlength n) (drop (1 − 1) (canrepr n)))⌋, Suc (length (numlist ns)) +
nlength n)
      using implant-contents nlcontents-def nllength-def by simp
    also have ... = (⌊numlist ns @ canrepr n⌋, Suc (length (numlist ns)) + nlength n)
      by simp
    also have ... = (⌊numlist ns @ canrepr n⌋, Suc (nllength ns) + nlength n)
      using nllength-def by simp
    also have ... = tps' ! j1
```

```
      by (metis assms(2) jk(1,2,4) nth-list-update-eq nth-list-update-neq)
    finally have implant (?tps ! j2) (?tps ! j1) (nlength n) = tps' ! j1 .
    then have tps' ! j1 = implant (?tps ! j2) (?tps ! j1) (nlength n)
      by simp
    then have tps' ! j1 = ?rhs ! j1
      by (simp add: jk)
    moreover have tps' ! j2 = ?rhs ! j2
      using assms(2) jk j2 by simp
    moreover have tps' ! j = ?rhs ! j if j < length tps' j ≠ j1 j ≠ j2 for j
      using that assms(2) by simp
    moreover have length tps' = length ?rhs
      using assms(2) by simp
    ultimately show ?thesis
      using nth-equalityI by blast
  qed
  show ttt = Suc (Suc (nllength ns) − i1) + Suc (nlength n)
    using assms(1) j1 tps0 i1 by simp
qed

definition tps3 ≡ tps0
    [j1 := (⌊numlist ns @ canrepr n⌋, Suc (nllength ns) + nlength n)]

lemma tm3 [transforms-intros]:
  assumes ttt = 6 + nllength ns − i1 + 2 ∗ nlength n
  shows transforms tm3 tps0 ttt tps3
  unfolding tm3-def
proof (tform tps: jk k)
  let ?tp1 = (⌊numlist ns @ canrepr n⌋, Suc (nllength ns) + nlength n)
  let ?tp2 = (⌊n⌋_N, Suc (nlength n))
  show clean-tape (tps0 [j1 := ?tp1, j2 := ?tp2] ! j2)
    by (simp add: clean-tape-ncontents jk)
  show tps3 = tps0[j1 := ?tp1, j2 := ?tp2, j2 := tps0 [j1 := ?tp1, j2 := ?tp2] ! j2 |#=| 1]
    using tps3-def tps0 jk
    by (metis (no-types, lifting) add-Suc fst-conv length-list-update list-update-id list-update-overwrite
      list-update-swap nth-list-update-eq)
  show ttt =
    3 + nllength ns − i1 + nlength n + (tps0[j1 := ?tp1, j2 := ?tp2] :#: j2 + 2)
  proof −
    have tps0[j1 := ?tp1, j2 := ?tp2] :#: j2 = Suc (nlength n)
      by (simp add: jk)
    then show ?thesis
      using jk tps0 i1 assms(1) by simp
  qed
qed

definition tps4 = tps0
    [j1 := (⌊numlist (ns @ [n])⌋, Suc (nllength (ns @ [n])))]

lemma tm4:
  assumes ttt = 7 + nllength ns − i1 + 2 ∗ nlength n
  shows transforms tm4 tps0 ttt tps4
  unfolding tm4-def
proof (tform tps: jk tps0 tps3-def)
  show ttt = 6 + nllength ns − i1 + 2 ∗ nlength n + 1
      using i1 assms(1) by simp
  show tps4 = tps3[j1 := tps3 ! j1 |:=| | |+| 1]
    (is tps4 = ?tps)
  proof −
    have ?tps = tps0[j1 := (⌊numlist ns @ canrepr n⌋, Suc (nllength ns + nlength n)) |:=| | |+| 1]
      using tps3-def by (simp add: jk)
    moreover have (⌊numlist ns @ canrepr n⌋, Suc (nllength ns + nlength n)) |:=| | |+| 1 =
        (⌊numlist (ns @ [n])⌋, Suc (nllength (ns @ [n])))
      (is ?lhs = ?rhs)
```

proof −
have *?lhs* = (⌊*numlist ns* @ *canrepr n*⌋(*Suc* (*nllength ns* + *nlength n*) := |), *Suc* (*Suc* (*nllength ns* + *nlength n*)))
by *simp*
also have ... = (⌊*numlist ns* @ *canrepr n*⌋(*Suc* (*nllength ns* + *nlength n*) := |), *Suc* (*nllength* (*ns* @ [*n*])))
using *nllength-def numlist-def* **by** *simp*
also have ... = (⌊(*numlist ns* @ *canrepr n*) @ [|]⌋, *Suc* (*nllength* (*ns* @ [*n*])))
using *contents-snoc* **by** (*metis length-append nllength-def*)
also have ... = (⌊*numlist ns* @ *canrepr n* @ [|]⌋, *Suc* (*nllength* (*ns* @ [*n*])))
by *simp*
also have ... = (⌊*numlist* (*ns* @ [*n*])⌋, *Suc* (*nllength* (*ns* @ [*n*])))
using *numlist-def* **by** *simp*
finally show *?lhs* = *?rhs* **.**
qed
ultimately show *?thesis*
using *tps4-def* **by** *auto*
qed
qed

end

end

lemma *tm-append* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx*
fixes *tps* :: *tape list* **and** *k i1 n* :: *nat* **and** *ns* :: *nat list*
assumes *0* < *j1*
assumes *length tps* = *k j1* < *k j2* < *k j1* ≠ *j2 i1* ≤ *Suc* (*nllength ns*)
and *tps* ! *j1* = (⌊*ns*⌋$_{NL}$, *i1*)
and *tps* ! *j2* = (⌊*n*⌋$_N$, *1*)
assumes *ttt* = *7* + *nllength ns* − *i1* + *2* ∗ *nlength n*
and *tps′* = *tps*
[*j1* := *nltape* (*ns* @ [*n*])]
shows *transforms* (*tm-append j1 j2*) *tps ttt tps′*
proof −
interpret *loc*: *turing-machine-append j1 j2* **.**
show *?thesis*
using *loc.tm4 loc.tm4-eq-tm-append loc.tps4-def assms nlcontents-def* **by** *simp*
qed

2.8.4 Computing the length

The next Turing machine counts the number of terminator symbols z on tape j_1 and stores the result on tape j_2. Thus, if j_1 contains a list of numbers, tape j_2 will contain the length of the list.

definition *tm-count* :: *tapeidx* ⇒ *tapeidx* ⇒ *symbol* ⇒ *machine* **where**
tm-count j1 j2 z ≡
WHILE tm-right-until j1 {□, *z*} ; λ*rs. rs* ! *j1* ≠ □ *DO*
tm-incr j2 ;;
tm-right j1
DONE ;;
tm-cr j1

lemma *tm-count-tm*:
assumes *0* < *j2* **and** *j1* < *k* **and** *j2* < *k* **and** *j1* ≠ *j2* **and** *G* ≥ *4*
shows *turing-machine k G* (*tm-count j1 j2 z*)
unfolding *tm-count-def*
using *turing-machine-loop-turing-machine tm-right-until-tm tm-incr-tm tm-cr-tm tm-right-tm assms*
by *simp*

locale *turing-machine-count* =
fixes *j1 j2* :: *tapeidx*
begin

definition *tmC* ≡ *tm-right-until j1* {□, |}
definition *tmB1* ≡ *tm-incr j2*
definition *tmB2* ≡ *tmB1* ;; *tm-right j1*
definition *tmL* ≡ *WHILE tmC* ; *λrs. rs* ! *j1* ≠ □ *DO tmB2 DONE*
definition *tm2* ≡ *tmL* ;; *tm-cr j1*

lemma *tm2-eq-tm-count*: *tm2* = *tm-count j1 j2* |
  unfolding *tmB1-def tmB2-def tmC-def tmL-def tm2-def tm-count-def*
  by *simp*

context
  fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *ns* :: *nat list*
  assumes *jk*: *j1* < *k j2* < *k 0* < *j2 j1* ≠ *j2 length tps0* = *k*
    and *tps0*:
      tps0 ! *j1* = (⌊*ns*⌋$_{NL}$, *1*)
      tps0 ! *j2* = (⌊*0*⌋$_N$, *1*)
begin

definition *tpsL t* ≡ *tps0*
  [*j1* := (⌊*ns*⌋$_{NL}$, *Suc* (*nllength* (*take t ns*))),
   j2 := (⌊*t*⌋$_N$, *1*)]

definition *tpsC t* ≡ *tps0*
  [*j1* := (⌊*ns*⌋$_{NL}$, *if t* < *length ns then nllength* (*take* (*Suc t*) *ns*) *else Suc* (*nllength ns*)),
   j2 := (⌊*t*⌋$_N$, *1*)]

lemma *tmC*:
  assumes *t* ≤ *length ns*
    and *ttt* = *Suc* (*if t* = *length ns then 0 else nlength* (*ns* ! *t*))
  shows *transforms tmC* (*tpsL t*) *ttt* (*tpsC t*)
  unfolding *tmC-def*
proof (*tform tps*: *jk tpsL-def tpsC-def*)
  let *?n* = *if t* = *length ns then 0 else nlength* (*ns* ! *t*)
  have ∗: *tpsL t* ! *j1* = (⌊*ns*⌋$_{NL}$, *Suc* (*nllength* (*take t ns*)))
    using *tpsL-def jk* **by** *simp*
  show *rneigh* (*tpsL t* ! *j1*) {□, |} *?n*
  proof (*cases t* = *length ns*)
    case *True*
    then have *tpsL t* ! *j1* = (⌊*ns*⌋$_{NL}$, *Suc* (*nllength ns*)) (**is** - = *?tp*)
      using ∗ **by** *simp*
    moreover from *this* **have** *fst ?tp* (*snd ?tp*) ∈ {□, |}
      by (*simp add*: *nlcontents-def nllength-def*)
    moreover have *?n* = *0*
      using *True* **by** *simp*
    ultimately show *?thesis*
      using *rneighI* **by** *simp*
  next
    case *False*
    then have *tpsL t* ! *j1* = (⌊*ns*⌋$_{NL}$, *Suc* (*nllength* (*take t ns*)))
      using ∗ **by** *simp*
    moreover have *?n* = *nlength* (*ns* ! *t*)
      using *False* **by** *simp*
    ultimately show *?thesis*
      using *nlcontents-rneigh-04* **by** (*simp add*: *False assms*(*1*) *le-neq-implies-less*)
  qed
  show *tpsC t* = (*tpsL t*) [*j1* := *tpsL t* ! *j1* |+| (*if t* = *length ns then 0 else nlength* (*ns* ! *t*))]
    (**is** *?l* = *?r*)
  proof (*rule nth-equalityI*)
    show *length ?l* = *length ?r*
      using *tpsC-def tpsL-def* **by** *simp*
    show *?l* ! *i* = *?r* ! *i* **if** *i* < *length ?l* **for** *i*
    proof −
      consider *i* = *j1* | *i* = *j2* | *i* ≠ *j1* ∧ *i* ≠ *j2*

        by *auto*
      then show *?thesis*
      proof (*cases*)
        case *1*
        show *?thesis*
        proof (*cases t = length ns*)
          case *True*
          then show *?thesis*
            using *1* **by** (*simp add*: *jk*(*2*,*4*) *tpsC-def tpsL-def*)
        next
          case *False*
          then have *t < length ns*
            using *assms*(*1*) **by** *simp*
          then show *?thesis*
            using *1 nllength-take-Suc jk tpsC-def tpsL-def* **by** *simp*
        qed
      next
        case *2*
        then show *?thesis*
          by (*simp add*: *jk*(*2*,*4*,*5*) *tpsC-def tpsL-def*)
      next
        case *3*
        then show *?thesis*
          by (*simp add*: *jk*(*2*,*4*) *tpsC-def tpsL-def*)
      qed
    qed
  qed
  show *ttt = Suc* (*if t = length ns then 0 else nlength* (*ns* ! *t*))
    using *assms*(*2*) **.**
qed

lemma *tmC′* [*transforms-intros*]:
  assumes *t ≤ length ns*
  shows *transforms tmC* (*tpsL t*) (*Suc* (*nllength ns*)) (*tpsC t*)
proof −
  have *Suc* (*if t = length ns then 0 else nlength* (*ns* ! *t*)) ≤ *Suc* (*if t = length ns then 0 else nllength ns*)
    using *assms member-le-nllength* **by** *simp*
  then have *Suc* (*if t = length ns then 0 else nlength* (*ns* ! *t*)) ≤ *Suc* (*nllength ns*)
    by *auto*
  then show *?thesis*
    using *tmC transforms-monotone assms* **by** *metis*
qed

definition *tpsB1 t ≡ tps0*
  [*j1* := (⌊*ns*⌋$_{NL}$, *nllength* (*take* (*Suc t*) *ns*)),
   j2 := (⌊*Suc t*⌋$_N$, *1*)]

lemma *tmB1* [*transforms-intros*]:
  assumes *t < length ns* **and** *ttt = 5 + 2 ∗ nlength t*
  shows *transforms tmB1* (*tpsC t*) *ttt* (*tpsB1 t*)
  unfolding *tmB1-def* **by** (*tform tps*: *jk tpsC-def tpsB1-def assms*)

definition *tpsB2 t ≡ tps0*
  [*j1* := (⌊*ns*⌋$_{NL}$, *Suc* (*nllength* (*take* (*Suc t*) *ns*))),
   j2 := (⌊*Suc t*⌋$_N$, *1*)]

lemma *tmB2*:
  assumes *t < length ns* **and** *ttt = 6 + 2 ∗ nlength t*
  shows *transforms tmB2* (*tpsC t*) *ttt* (*tpsB2 t*)
  unfolding *tmB2-def* **by** (*tform tps*: *jk tpsB1-def tpsB2-def assms*)

lemma *tpsB2-eq-tpsL*: *tpsB2 t = tpsL* (*Suc t*)
  using *tpsB2-def tpsL-def* **by** *simp*

lemma *tmB2'* [*transforms-intros*]:
assumes $t <$ *length ns*
shows *transforms tmB2* (*tpsC t*) ($6 + 2 *$ *nllength ns*) (*tpsL* (*Suc t*))
proof −
have *nlength t* $\leq$ *nllength ns*
using *assms*(*1*) *length-le-nllength nlength-le-n* **by** (*meson le-trans less-or-eq-imp-le*)
then have $6 + 2 *$ *nlength t* $\leq 6 + 2 *$ *nllength ns*
by *simp*
then show *?thesis*
using *assms tmB2 transforms-monotone tpsB2-eq-tpsL* **by** *metis*
qed

lemma *tmL* [*transforms-intros*]:
assumes *ttt* $= 13 *$ *nllength ns* $\hat{}\ 2 + 2$
shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsC* (*length ns*))
unfolding *tmL-def*
proof (*tform*)
show *read* (*tpsC t*) ! *j1* $\neq \Box$ **if** $t <$ *length ns* **for** *t*
proof −
have *tpsC t* ! *j1* = ($\lfloor ns \rfloor_{NL}$, *if* $t <$ *length ns then nllength* (*take* (*Suc t*) *ns*) *else Suc* (*nllength ns*))
using *tpsC-def jk* **by** *simp*
then have ∗: *tpsC t* ! *j1* = ($\lfloor ns \rfloor_{NL}$, *nllength* (*take* (*Suc t*) *ns*)) (**is** - = *?tp*)
using *that* **by** *simp*
have *fst ?tp* (*snd ?tp*) = $\lfloor ns \rfloor_{NL}$ (*nllength* (*take* (*Suc t*) *ns*))
by *simp*
also have ... = $\lfloor$*numlist ns*$\rfloor$ (*nllength* (*take* (*Suc t*) *ns*))
using *nlcontents-def* **by** *simp*
also have ... = *numlist ns* ! (*nllength* (*take* (*Suc t*) *ns*) − *1*)
using *nllength that contents-inbounds nllength-def nllength-take nllength-take-Suc*
by (*metis Suc-leI add-Suc-right less-nat-zero-code not-less-eq*)
also have ... = *4*
using *numlist-take-Suc-at-term*[*OF that*] **by** *simp*
finally have *fst ?tp* (*snd ?tp*) = | **.**
then have *fst ?tp* (*snd ?tp*) $\neq \Box$
by *simp*
then show *?thesis*
using ∗ *jk*(*1*,*5*) *length-list-update tapes-at-read' tpsC-def* **by** *metis*
qed
show ¬ *read* (*tpsC* (*length ns*)) ! *j1* $\neq \Box$
proof −
have *tpsC* (*length ns*) ! *j1* = ($\lfloor ns \rfloor_{NL}$, *Suc* (*nllength ns*)) (**is** - = *?tp*)
using *tpsC-def jk* **by** *simp*
moreover have *fst ?tp* (*snd ?tp*) = *0*
by (*simp add*: *nlcontents-def nllength-def*)
ultimately have *read* (*tpsC* (*length ns*)) ! *j1* = $\Box$
using *jk*(*1*,*5*) *length-list-update tapes-at-read' tpsC-def* **by** *metis*
then show *?thesis*
by *simp*
qed
show *length ns* ∗ (*Suc* (*nllength ns*) + ($6 + 2 *$ *nllength ns*) + *2*) + *Suc* (*nllength ns*) + *1* $\leq$ *ttt*
proof −
have *length ns* ∗ (*Suc* (*nllength ns*) + ($6 + 2 *$ *nllength ns*) + *2*) + *Suc* (*nllength ns*) + *1* =
length ns ∗ ($9 + 3 *$ *nllength ns*) + *nllength ns* + *2*
by *simp*
also have ... $\leq$ *nllength ns* ∗ ($9 + 3 *$ *nllength ns*) + *nllength ns* + *2*
by (*simp add*: *length-le-nllength*)
also have ... = *nllength ns* ∗ ($10 + 3 *$ *nllength ns*) + *2*
by *algebra*
also have ... = $10 *$ *nllength ns* + $3 *$ *nllength ns* $\hat{}\ 2 + 2$
by *algebra*
also have ... $\leq 10 *$ *nllength ns* $\hat{}\ 2 + 3 *$ *nllength ns* $\hat{}\ 2 + 2$
by (*meson add-mono-thms-linordered-semiring*(*1*) *le-eq-less-or-eq mult-le-mono2 power2-nat-le-imp-le*)

also have ... = *13* ∗ *nllength ns* ^ *2* + *2*
by *simp*
finally show *?thesis*
using *assms* **by** *simp*
qed
qed

definition *tps2* ≡ *tps0*
[*j2* := (⌊*length ns*⌋$_N$, *1*)]

lemma *tm2*:
assumes *ttt* = *13* ∗ *nllength ns* ^ *2* + *5* + *nllength ns*
shows *transforms tm2* (*tpsL 0*) *ttt tps2*
unfolding *tm2-def*
proof (*tform tps*: *jk tpsC-def tps2-def*)
have ∗: *tpsC* (*length ns*) ! *j1* = (⌊*ns*⌋$_{NL}$, *Suc* (*nllength ns*))
using *jk tpsC-def* **by** *simp*
then show *clean-tape* (*tpsC* (*length ns*) ! *j1*)
by (*simp add*: *clean-tape-nlcontents*)
show *tps2* = (*tpsC* (*length ns*))[*j1* := *tpsC* (*length ns*) ! *j1* |#=| *1*]
using *jk tps0*(*1*) *tps2-def tpsC-def* ∗ **by** (*metis fstI list-update-id list-update-overwrite list-update-swap*)
show *ttt* = *13* ∗ (*nllength ns*)^2 + *2* + (*tpsC* (*length ns*) :#: *j1* + *2*)
using *assms* ∗ **by** *simp*
qed

lemma *tpsL-eq-tps0*: *tpsL 0* = *tps0*
using *tpsL-def tps0* **by** (*metis One-nat-def list-update-id nllength-Nil take0*)

lemma *tm2'*:
assumes *ttt* = *14* ∗ *nllength ns* ^ *2* + *5*
shows *transforms tm2 tps0 ttt tps2*
proof −
have *nllength ns* ≤ *nllength ns* ^ *2*
using *power2-nat-le-imp-le* **by** *simp*
then have *13* ∗ *nllength ns* ^ *2* + *5* + *nllength ns* ≤ *ttt*
using *assms* **by** *simp*
then show *?thesis*
using *assms tm2 transforms-monotone tpsL-eq-tps0* **by** *simp*
qed

end

end

lemma *transforms-tm-count-4I* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx*
fixes *tps tps'* :: *tape list* **and** *k* :: *nat* **and** *ns* :: *nat list*
assumes *j1* < *k j2* < *k 0* < *j2 j1* ≠ *j2 length tps* = *k*
assumes
tps ! *j1* = (⌊*ns*⌋$_{NL}$, *1*)
tps ! *j2* = (⌊*0*⌋$_N$, *1*)
assumes *ttt* = *14* ∗ *nllength ns* ^ *2* + *5*
assumes *tps'* = *tps*[*j2* := (⌊*length ns*⌋$_N$, *1*)]
shows *transforms* (*tm-count j1 j2* |) *tps ttt tps'*
proof −
interpret *loc*: *turing-machine-count j1 j2* **.**
show *?thesis*
using *loc.tm2-eq-tm-count loc.tm2' loc.tps2-def assms* **by** *simp*
qed

2.8.5 Extracting the n-th element

The next Turing machine expects a list on tape j_1 and an index i on j_2 and writes the i-th element of the list to j_2, overwriting i. The TM does not terminate for out-of-bounds access, which of course we will never attempt. Again the parameter z is a generic terminator symbol.

```
definition tm-nth-inplace :: tapeidx ⇒ tapeidx ⇒ symbol ⇒ machine where
  tm-nth-inplace j1 j2 z ≡
    WHILE [] ; λrs. rs ! j2 ≠ □ DO
      tm-decr j2 ;;
      tm-right-until j1 {z} ;;
      tm-right j1
    DONE ;;
    tm-cp-until j1 j2 {z} ;;
    tm-cr j2 ;;
    tm-cr j1

lemma tm-nth-inplace-tm:
  assumes k ≥ 2 and G ≥ 4 and 0 < j2 and j1 < k and j2 < k
  shows turing-machine k G (tm-nth-inplace j1 j2 |)
  unfolding tm-nth-inplace-def
  using assms tm-decr-tm tm-right-until-tm tm-right-tm tm-cp-until-tm tm-cr-tm Nil-tm
  by (simp add: assms(1) turing-machine-loop-turing-machine)

locale turing-machine-nth-inplace =
  fixes j1 j2 :: tapeidx
begin

definition tmL1 ≡ tm-decr j2
definition tmL2 ≡ tmL1 ;; tm-right-until j1 {|}
definition tmL3 ≡ tmL2 ;; tm-right j1
definition tmL ≡ WHILE [] ; λrs. rs ! j2 ≠ □ DO tmL3 DONE
definition tm2 ≡ tmL ;; tm-cp-until j1 j2 {|}
definition tm3 ≡ tm2 ;; tm-cr j2
definition tm4 ≡ tm3 ;; tm-cr j1

lemma tm4-eq-tm-nth-inplace: tm4 = tm-nth-inplace j1 j2 |
  unfolding tmL1-def tmL2-def tmL3-def tmL-def tm2-def tm3-def tm4-def tm-nth-inplace-def
  by simp

context
  fixes tps0 :: tape list and k idx :: nat and ns :: nat list
  assumes jk: j1 < k j2 < k 0 < j2 j1 ≠ j2 length tps0 = k
    and idx: idx < length ns
    and tps0:
      tps0 ! j1 = (⌊ns⌋_NL, 1)
      tps0 ! j2 = (⌊idx⌋_N, 1)
begin

definition tpsL t ≡ tps0
  [j1 := (⌊ns⌋_NL, Suc (nllength (take t ns))),
   j2 := (⌊idx − t⌋_N, 1)]

definition tpsL1 t ≡ tps0
  [j1 := (⌊ns⌋_NL, Suc (nllength (take t ns))),
   j2 := (⌊idx − t − 1⌋_N, 1)]

lemma tmL1 [transforms-intros]:
  assumes ttt = 8 + 2 ∗ nlength (idx − t)
  shows transforms tmL1 (tpsL t) ttt (tpsL1 t)
  unfolding tmL1-def by (tform tps: tpsL-def tpsL1-def jk time: assms)

definition tpsL2 t ≡ tps0
  [j1 := (⌊ns⌋_NL, nllength (take (Suc t) ns)),
```

```
  j2 := (⌊idx − t − 1⌋_N, 1)]

lemma tmL2:
  assumes t < length ns and ttt = 8 + 2 ∗ nlength (idx − t) + Suc (nlength (ns ! t))
  shows transforms tmL2 (tpsL t) ttt (tpsL2 t)
  unfolding tmL2-def
proof (tform tps: jk tpsL1-def tpsL2-def time: assms(2))
  let ?l = nlength (ns ! t)
  show rneigh (tpsL1 t ! j1) {|} ?l
  proof −
    have tpsL1 t ! j1 = (⌊ns⌋_NL, Suc (nllength (take t ns)))
      using tpsL1-def jk by (simp only: nth-list-update-eq nth-list-update-neq)
    then show ?thesis
      using assms(1) nlcontents-rneigh-4 by simp
  qed
  show tpsL2 t = (tpsL1 t)[j1 := tpsL1 t ! j1 |+| nlength (ns ! t)]
    (is ?l = ?r)
  proof (rule nth-equalityI)
    show len: length ?l = length ?r
      using tpsL1-def tpsL2-def jk by simp
    show ?l ! i = ?r ! i if i < length ?l for i
    proof −
      consider i = j1 | i = j2 | i ≠ j1 ∧ i ≠ j2
        by auto
      then show ?thesis
      proof (cases)
        case 1
        then show ?thesis
          using that tpsL1-def tpsL2-def jk nllength-take-Suc[OF assms(1)] by simp
      next
        case 2
        then show ?thesis
          using that tpsL1-def tpsL2-def jk
          by (simp only: nth-list-update-eq nth-list-update-neq' length-list-update)
      next
        case 3
        then show ?thesis
          using that tpsL1-def tpsL2-def jk
          by (simp only: nth-list-update-eq jk nth-list-update-neq' length-list-update)
      qed
    qed
  qed
qed

lemma tmL2' [transforms-intros]:
  assumes t < length ns and ttt = 9 + 2 ∗ nlength idx + nlength (Max (set ns))
  shows transforms tmL2 (tpsL t) ttt (tpsL2 t)
proof −
  let ?ttt = 8 + 2 ∗ nlength (idx − t) + Suc (nlength (ns ! t))
  have transforms tmL2 (tpsL t) ?ttt (tpsL2 t)
    using tmL2 assms by simp
  moreover have ttt ≥ ?ttt
  proof −
    have nlength (idx − t) ≤ nlength idx
      using nlength-mono by simp
    moreover have nlength (ns ! t) ≤ nlength (Max (set ns))
      using nlength-mono assms by simp
    ultimately show ?thesis
      using assms(2) by simp
  qed
  ultimately show ?thesis
    using transforms-monotone by simp
qed
```

definition *tpsL3 t* ≡ *tps0*
  [*j1* := (⌊*ns*⌋$_{NL}$, *Suc* (*nllength* (*take* (*Suc t*) *ns*))),
   j2 := (⌊*idx* − *t* − *1*⌋$_N$, *1*)]

lemma *tmL3*:
  assumes *t* < *length ns* **and** *ttt* = *10* + *2* ∗ *nlength idx* + *nlength* (*Max* (*set ns*))
  shows *transforms tmL3* (*tpsL t*) *ttt* (*tpsL3 t*)
  unfolding *tmL3-def*
proof (*tform tps*: *jk tpsL2-def tpsL3-def assms*(*1*) *time*: *assms*(*2*))
  show *tpsL3 t* = (*tpsL2 t*)[*j1* := *tpsL2 t* ! *j1* |+| *1*]
    using *tpsL3-def tpsL2-def jk tps0*
    by (*metis Groups.add-ac*(*2*) *fst-conv list-update-overwrite list-update-swap nth-list-update nth-list-update-neq plus-1-eq-Suc snd-conv*)
qed

lemma *tpsL3-eq-tpsL*: *tpsL3 t* = *tpsL* (*Suc t*)
  using *tpsL3-def tpsL-def* **by** *simp*

lemma *tmL*:
  assumes *ttt* = *idx* ∗ (*12* + *2* ∗ *nlength idx* + *nlength* (*Max* (*set ns*))) + *1*
  shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsL idx*)
  unfolding *tmL-def*
proof (*tform*)
  let *?t* = *10* + *2* ∗ *nlength idx* + *nlength* (*Max* (*set ns*))
  show ⋀*t*. *t* < *idx* ⟹ *transforms tmL3* (*tpsL t*) *?t* (*tpsL* (*Suc t*))
    using *tmL3 tpsL3-eq-tpsL idx* **by** *simp*
  show *read* (*tpsL t*) ! *j2* ≠ □ **if** *t* < *idx* **for** *t*
  proof −
    have *tpsL t* ! *j2* = (⌊*idx* − *t*⌋$_N$, *1*)
      using *tpsL-def jk* **by** *simp*
    then have *read* (*tpsL t*) ! *j2* = ⌊*idx* − *t*⌋$_N$ *1*
      using *tapes-at-read′ jk tpsL-def* **by** (*metis fst-conv length-list-update snd-conv*)
    moreover have *idx* − *t* > *0*
      using *that* **by** *simp*
    ultimately show *read* (*tpsL t*) ! *j2* ≠ □
      using *ncontents-1-blank-iff-zero* **by** *simp*
  qed
  show ¬ *read* (*tpsL idx*) ! *j2* ≠ □
  proof −
    have *tpsL idx* ! *j2* = (⌊*idx* − *idx*⌋$_N$, *1*)
      using *tpsL-def jk* **by** *simp*
    then have *read* (*tpsL idx*) ! *j2* = ⌊*idx* − *idx*⌋$_N$ *1*
      using *tapes-at-read′ jk tpsL-def* **by** (*metis fst-conv length-list-update snd-conv*)
    then show *?thesis*
      using *ncontents-1-blank-iff-zero* **by** *simp*
  qed
  show *idx* ∗ (*10* + *2* ∗ *nlength idx* + *nlength* (*Max* (*set ns*)) + *2*) + *1* ≤ *ttt*
    using *assms* **by** *simp*
qed

definition *tps1* ≡ *tps0*
  [*j1* := (⌊*ns*⌋$_{NL}$, *Suc* (*nllength* (*take idx ns*))),
   j2 := (⌊*0*⌋$_N$, *1*)]

lemma *tps1-eq-tpsL*: *tps1* = *tpsL idx*
  using *tps1-def tpsL-def* **by** *simp*

lemma *tps0-eq-tpsL*: *tps0* = *tpsL 0*
  using *tps0 tpsL-def nllength-Nil* **by** (*metis One-nat-def list-update-id minus-nat.diff-0 take0*)

lemma *tmL′* [*transforms-intros*]:
  assumes *ttt* = *idx* ∗ (*12* + *2* ∗ *nlength idx* + *nlength* (*Max* (*set ns*))) + *1*

```
  shows transforms tmL tps0 ttt tps1
  using tmL assms tps0-eq-tpsL tps1-eq-tpsL by simp

definition tps2 ≡ tps0
  [j1 := (⌊ns⌋NL, nllength (take (Suc idx) ns)),
   j2 := (⌊ns ! idx⌋N, Suc (nlength (ns ! idx)))]

lemma tm2 [transforms-intros]:
  assumes ttt = idx * (12 + 2 * nlength idx + nlength (Max (set ns))) + 2 + nlength (ns ! idx)
  shows transforms tm2 tps0 ttt tps2
  unfolding tm2-def
proof (tform tps: jk tps2-def tps1-def time: assms)
  have tps1 ! j1 = (⌊ns⌋NL, Suc (nllength (take idx ns)))
    using tps1-def tps0 jk by simp
  then show rneigh (tps1 ! j1) {|} (nlength (ns ! idx))
    by (simp add: idx nlcontents-rneigh-4)
  show tps2 = tps1
    [j1 := tps1 ! j1 |+| nlength (ns ! idx),
     j2 := implant (tps1 ! j1) (tps1 ! j2) (nlength (ns ! idx))]
     (is ?l = ?r)
  proof (rule nth-equalityI)
    show len: length ?l = length ?r
      using tps1-def tps2-def by simp
    show ?l ! i = ?r ! i if i < length ?l for i
    proof -
      consider i = j1 | i = j2 | i ≠ j1 ∧ i ≠ j2
        by auto
      then show ?thesis
      proof (cases)
        case 1
        then show ?thesis
          using that tps1-def tps2-def jk nllength-take-Suc idx by simp
      next
        case 2
        then have lhs: ?l ! i = (⌊ns ! idx⌋N, Suc (nlength (ns ! idx)))
          using tps2-def jk by simp
        let ?i = Suc (nllength (take idx ns))
        have i1: ?i > 0
          by simp
        have nlength (ns ! idx) + (?i − 1) ≤ nllength ns
          using idx by (simp add: add.commute less-or-eq-imp-le nllength-take)
        then have i2: nlength (ns ! idx) + (?i − 1) ≤ length (numlist ns)
          using nllength-def by simp
        have ?r ! i = implant (tps1 ! j1) (tps1 ! j2) (nlength (ns ! idx))
          using 2 tps1-def jk by simp
        also have ... = implant (⌊ns⌋NL, ?i) (⌊0⌋N, 1) (nlength (ns ! idx))
        proof -
          have tps1 ! j1 = (⌊ns⌋NL, Suc (nllength (take idx ns)))
            using tps1-def jk by simp
          moreover have tps1 ! j2 = (⌊0⌋N, 1)
            using tps1-def jk by simp
          ultimately show ?thesis
            by simp
        qed
        also have ... = (⌊[] @ (take (nlength (ns ! idx)) (drop (?i − 1) (numlist ns)))⌋, Suc (length []) + nlength
(ns ! idx))
          using implant-contents[OF i1 i2] by (metis One-nat-def list.size(3) ncontents-0 nlcontents-def)
        finally have ?r ! i = (⌊[] @ (take (nlength (ns ! idx)) (drop (?i − 1) (numlist ns)))⌋, Suc (length []) +
nlength (ns ! idx)) .
        then have ?r ! i = (⌊take (nlength (ns ! idx)) (drop (nllength (take idx ns)) (numlist ns))⌋, Suc (nlength
(ns ! idx)))
          by simp
        then have ?r ! i = (⌊canrepr (ns ! idx)⌋, Suc (nlength (ns ! idx)))
```

```
        using take-drop-numlist'[OF idx] by simp
      then show ?thesis
        using lhs by simp
    next
      case 3
      then show ?thesis
        using that tps1-def tps2-def jk by simp
    qed
  qed
 qed
qed

definition tps3 ≡ tps0
  [j1 := (⌊ns⌋NL, nllength (take (Suc idx) ns)),
   j2 := (⌊ns ! idx⌋N, 1)]

lemma tm3 [transforms-intros]:
  assumes ttt = idx * (12 + 2 * nlength idx + nlength (Max (set ns))) + 5 + 2 * nlength (ns ! idx)
  shows transforms tm3 tps0 ttt tps3
  unfolding tm3-def
  by (tform tps: clean-tape-ncontents jk tps2-def tps3-def time: assms tps2-def jk)

definition tps4 ≡ tps0
  [j2 := (⌊ns ! idx⌋N, 1)]

lemma tm4:
  assumes ttt = idx * (12 + 2 * nlength idx + nlength (Max (set ns))) + 7 + 2 * nlength (ns ! idx) + nllength
(take (Suc idx) ns)
  shows transforms tm4 tps0 ttt tps4
  unfolding tm4-def
proof (tform tps: clean-tape-nlcontents jk tps3-def tps4-def time: assms jk tps3-def)
  show tps4 = tps3[j1 := tps3 ! j1 |#=| 1]
    using tps4-def tps3-def jk tps0(1) list-update-id[of tps0 j1] by (simp add: list-update-swap)
qed

lemma tm4':
  assumes ttt = 18 * nllength ns ^ 2 + 12
  shows transforms tm4 tps0 ttt tps4
proof -
  let ?ttt = idx * (12 + 2 * nlength idx + nlength (Max (set ns))) + 7 + 2 * nlength (ns ! idx) + nllength
(take (Suc idx) ns)

  have 1: idx ≤ nllength ns
    using idx length-le-nllength by (meson le-trans less-or-eq-imp-le)
  then have 2: nlength idx ≤ nllength ns
    using nlength-mono nlength-le-n by (meson dual-order.trans)
  have ns ≠ []
    using idx by auto
  then have 3: nlength (Max (set ns)) ≤ nllength ns
    using member-le-nllength by simp
  have 4: nlength (ns ! idx) ≤ nllength ns
    using idx member-le-nllength by simp
  have 5: nllength (take (Suc idx) ns) ≤ nllength ns
    by (metis Suc-le-eq add-Suc-right idx nllength-take nllength-take-Suc)

  have ?ttt ≤ idx * (12 + 2 * nllength ns + nllength ns) + 7 + 2 * nllength ns + nllength ns
    using 2 3 4 5 by (meson add-le-mono le-eq-less-or-eq mult-le-mono2)
  also have ... = idx * (12 + 3 * nllength ns) + 7 + 3 * nllength ns
    by simp
  also have ... ≤ idx * (12 + 3 * nllength ns) + (12 + 3 * nllength ns)
    by simp
  also have ... = Suc idx * (12 + 3 * nllength ns)
    by simp
```

also have ... $\leq$ *Suc* (*nllength ns*) $*$ (*12* + *3* $*$ *nllength ns*)
  using *1* **by** *simp*
also have ... = *nllength ns* $*$ (*12* + *3* $*$ *nllength ns*) + (*12* + *3* $*$ *nllength ns*)
  by *simp*
also have ... = *12* $*$ *nllength ns* + *3* $*$ *nllength ns* ^ *2* + *12* + *3* $*$ *nllength ns*
  by *algebra*
also have ... = *15* $*$ *nllength ns* + *3* $*$ *nllength ns* ^ *2* + *12*
  by *simp*
also have ... $\leq$ *18* $*$ *nllength ns* ^ *2* + *12*
  by (*simp add*: *power2-eq-square*)
finally have *?ttt* $\leq$ *18* $*$ *nllength ns* ^ *2* + *12* **.**
then show *?thesis*
  using *tm4 transforms-monotone assms* **by** *simp*
qed

end

end

lemma *transforms-tm-nth-inplace-4I* [*transforms-intros*]:
  fixes *j1 j2* :: *tapeidx*
  fixes *tps tps'* :: *tape list* **and** *k idx* :: *nat* **and** *ns* :: *nat list*
  assumes *j1* < *k j2* < *k 0* < *j2 j1* $\neq$ *j2 length tps* = *k*
    and *idx* < *length ns*
  assumes
    tps ! *j1* = ($\lfloor ns \rfloor_{NL}$, *1*)
    tps ! *j2* = ($\lfloor idx \rfloor_{N}$, *1*)
  assumes *ttt* = *18* $*$ *nllength ns* ^ *2* + *12*
  assumes *tps'* = *tps*
    [*j2* := ($\lfloor ns\ !\ idx \rfloor_{N}$, *1*)]
  shows *transforms* (*tm-nth-inplace j1 j2* |) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-nth-inplace j1 j2* **.**
  show *?thesis*
    using *assms loc.tm4-eq-tm-nth-inplace loc.tm4' loc.tps4-def* **by** *simp*
qed

The next Turing machine expects a list on tape j_1 and an index i on tape j_2. It writes the i-th element of the list to tape j_3. Like the previous TM, it will not terminate on out-of-bounds access, and z is a parameter for the symbol that terminates the list elements.

definition *tm-nth* :: *tapeidx* $\Rightarrow$ *tapeidx* $\Rightarrow$ *tapeidx* $\Rightarrow$ *symbol* $\Rightarrow$ *machine* **where**
  tm-nth j1 j2 j3 z $\equiv$
    tm-copyn j2 j3 ;;
    tm-nth-inplace j1 j3 z

lemma *tm-nth-tm*:
  assumes *k* $\geq$ *2* **and** *G* $\geq$ *4* **and** *0* < *j2 0* < *j1 j1* < *k j2* < *k 0* < *j3 j3* < *k j2* $\neq$ *j3*
  shows *turing-machine k G* (*tm-nth j1 j2 j3* |)
  unfolding *tm-nth-def* **using** *tm-copyn-tm tm-nth-inplace-tm assms* **by** *simp*

lemma *transforms-tm-nth-4I* [*transforms-intros*]:
  fixes *j1 j2 j3* :: *tapeidx*
  fixes *tps tps'* :: *tape list* **and** *k idx* :: *nat* **and** *ns* :: *nat list*
  assumes *j1* < *k j2* < *k j3* < *k 0* < *j1 0* < *j2 0* < *j3 j1* $\neq$ *j2 j2* $\neq$ *j3 j1* $\neq$ *j3*
    and *length tps* = *k*
    and *idx* < *length ns*
  assumes
    tps ! *j1* = ($\lfloor ns \rfloor_{NL}$, *1*)
    tps ! *j2* = ($\lfloor idx \rfloor_{N}$, *1*)
    tps ! *j3* = ($\lfloor 0 \rfloor_{N}$, *1*)
  assumes *ttt* = *21* $*$ *nllength ns* ^ *2* + *26*
  assumes *tps'* = *tps*
    [*j3* := ($\lfloor ns\ !\ idx \rfloor_{N}$, *1*)]

```
  shows transforms (tm-nth j1 j2 j3 |) tps ttt tps'
proof -
  let ?ttt = 14 + 3 * (nlength idx + nlength 0) + (18 * (nllength ns)^2 + 12)
  have transforms (tm-nth j1 j2 j3 |) tps ?ttt tps'
    unfolding tm-nth-def
  proof (tform tps: assms(1−11))
    show tps ! j2 = (⌊idx⌋_N, 1)
      using assms by simp
    show tps ! j3 = (⌊0⌋_N, 1)
      using assms by simp
    show tps[j3 := (⌊idx⌋_N, 1)] ! j1 = (⌊ns⌋_NL, 1)
      using assms by simp
    then show tps' = tps
      [j3 := (⌊idx⌋_N, 1),
       j3 := (⌊ns ! idx⌋_N, 1)]
      using assms by (metis One-nat-def list-update-overwrite)
  qed
  moreover have ?ttt ≤ ttt
  proof -
    have nlength idx ≤ idx
      using nlength-le-n by simp
    then have nlength idx ≤ length ns
      using assms(11) by simp
    then have nlength idx ≤ nllength ns
      using length-le-nllength by (meson order.trans)
    then have nlength idx ≤ nllength ns ^ 2
      by (meson le-refl order-trans power2-nat-le-imp-le)
    moreover have ?ttt = 3 * nlength idx + 18 * (nllength ns)^2 + 26
      by simp
    ultimately show ?thesis
      using assms(15) by simp
  qed
  ultimately show ?thesis
    using transforms-monotone by simp
qed
```

2.8.6 Finding the previous position of an element

The Turing machine in this section implements a slightly peculiar functionality, which we will start using only in Section 6. Given a list of numbers and an index i it determines the greatest index less than i where the list contains the same element as in position i. If no such element exists, it returns i. Formally:

```
definition previous :: nat list ⇒ nat ⇒ nat where
  previous ns idx ≡
    if ∃ i<idx. ns ! i = ns ! idx
    then GREATEST i. i < idx ∧ ns ! i = ns ! idx
    else idx

lemma previous-less:
  assumes ∃ i<idx. ns ! i = ns ! idx
  shows previous ns idx < idx ∧ ns ! (previous ns idx) = ns ! idx
proof -
  let ?P = λi. i < idx ∧ ns ! i = ns ! idx
  have previous ns idx = (GREATEST i. ?P i)
    using assms previous-def by simp
  moreover have ∀ y. ?P y ⟶ y ≤ idx
    by simp
  ultimately show ?thesis
    using GreatestI-ex-nat[OF assms, of idx] by simp
qed

lemma previous-eq: previous ns idx = idx ⟷ ¬ (∃ i<idx. ns ! i = ns ! idx)
  using previous-def nat-less-le previous-less by simp
```

lemma *previous-le*: *previous ns idx* ≤ *idx*
  using *previous-eq previous-less* **by** (*metis less-or-eq-imp-le*)

The following Turing machine expects the list on tape j_1 and the index on tape j_2. It outputs the result on tape $j_2 + 5$. The tapes $j_2 + 1, \ldots, j_2 + 4$ are scratch space.

```
definition tm-prev :: tapeidx ⇒ tapeidx ⇒ machine where
  tm-prev j1 j2 ≡
    tm-copyn j2 (j2 + 5) ;;
    tm-nth j1 j2 (j2 + 1) | ;;
    WHILE tm-equalsn (j2 + 2) j2 (j2 + 4) ; λrs. rs ! (j2 + 4) = □ DO
      tm-nth j1 (j2 + 2) (j2 + 3) | ;;
      tm-equalsn (j2 + 1) (j2 + 3) (j2 + 4) ;;
      tm-setn (j2 + 3) 0 ;;
      IF λrs. rs ! (j2 + 4) ≠ □ THEN
        tm-setn (j2 + 4) 0 ;;
        tm-copyn (j2 + 2) (j2 + 5)
      ELSE
        []
      ENDIF ;;
      tm-incr (j2 + 2)
    DONE ;;
    tm-erase-cr (j2 + 1) ;;
    tm-erase-cr (j2 + 2) ;;
    tm-erase-cr (j2 + 4)

lemma tm-prev-tm:
  assumes k ≥ j2 + 6 and G ≥ 4 and j1 < j2 and 0 < j1
  shows turing-machine k G (tm-prev j1 j2)
  unfolding tm-prev-def
  using assms tm-copyn-tm tm-nth-tm tm-equalsn-tm tm-setn-tm tm-incr-tm Nil-tm tm-erase-cr-tm
    turing-machine-loop-turing-machine turing-machine-branch-turing-machine
  by simp

locale turing-machine-prev =
  fixes j1 j2 :: tapeidx
begin

definition tm1 ≡ tm-copyn j2 (j2 + 5)
definition tm2 ≡ tm1 ;; tm-nth j1 j2 (j2 + 1) |
definition tmC ≡ tm-equalsn (j2 + 2) j2 (j2 + 4)
definition tmB1 ≡ tm-nth j1 (j2 + 2) (j2 + 3) |
definition tmB2 ≡ tmB1 ;; tm-equalsn (j2 + 1) (j2 + 3) (j2 + 4)
definition tmB3 ≡ tmB2 ;; tm-setn (j2 + 3) 0
definition tmI1 ≡ tm-setn (j2 + 4) 0
definition tmI2 ≡ tmI1 ;; tm-copyn (j2 + 2) (j2 + 5)
definition tmI ≡ IF λrs. rs ! (j2 + 4) ≠ □ THEN tmI2 ELSE [] ENDIF
definition tmB4 ≡ tmB3 ;; tmI
definition tmB5 ≡ tmB4 ;; tm-incr (j2 + 2)
definition tmL ≡ WHILE tmC ; λrs. rs ! (j2 + 4) = □ DO tmB5 DONE
definition tm3 ≡ tm2 ;; tmL
definition tm4 ≡ tm3 ;; tm-erase-cr (j2 + 1)
definition tm5 ≡ tm4 ;; tm-erase-cr (j2 + 2)
definition tm6 ≡ tm5 ;; tm-erase-cr (j2 + 4)

lemma tm6-eq-tm-prev: tm6 = tm-prev j1 j2
  unfolding tm-prev-def tm3-def tmL-def tmB5-def tmB4-def tmI-def tmI2-def tmI1-def tmB3-def
    tmB2-def tmB1-def tmC-def tm2-def tm1-def tm4-def tm5-def tm6-def
  by simp

context
  fixes tps0 :: tape list and k idx :: nat and ns :: nat list
  assumes jk: 0 < j1 j1 < j2 j2 + 6 ≤ k length tps0 = k
    and idx: idx < length ns
```

and *tps0*:
tps0 ! *j1* = ($\lfloor ns \rfloor_{NL}$, *1*)
tps0 ! *j2* = ($\lfloor idx \rfloor_N$, *1*)
tps0 ! (*j2* + *1*) = ($\lfloor 0 \rfloor_N$, *1*)
tps0 ! (*j2* + *2*) = ($\lfloor 0 \rfloor_N$, *1*)
tps0 ! (*j2* + *3*) = ($\lfloor 0 \rfloor_N$, *1*)
tps0 ! (*j2* + *4*) = ($\lfloor 0 \rfloor_N$, *1*)
tps0 ! (*j2* + *5*) = ($\lfloor 0 \rfloor_N$, *1*)
begin

definition *tps1* ≡ *tps0*
[*j2* + *5* := ($\lfloor idx \rfloor_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *14* + *3* ∗ *nlength idx*
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def* **by** (*tform tps*: *jk idx tps0 tps1-def assms nlength-0*)

definition *tps2* ≡ *tps0*
[*j2* + *1* := ($\lfloor ns ! idx \rfloor_N$, *1*),
j2 + *5* := ($\lfloor idx \rfloor_N$, *1*)]

lemma *tm2*:
assumes *ttt* = *14* + *3* ∗ *nlength idx* + (*21* ∗ (*nllength ns*)^2 + *26*)
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def* **by** (*tform tps*: *jk idx tps0 tps1-def tps2-def time*: *assms*)

definition *rv* :: *nat* ⇒ *nat* **where**
rv t ≡ *if* ∃ *i*<*t*. *ns* ! *i* = *ns* ! *idx then GREATEST i*. *i* < *t* ∧ *ns* ! *i* = *ns* ! *idx else idx*

lemma *rv-0*: *rv 0* = *idx*
using *rv-def* **by** *simp*

lemma *rv-le*: *rv t* ≤ *max t idx*
proof (*cases* ∃ *i*<*t*. *ns* ! *i* = *ns* ! *idx*)
case *True*
let *?Q* = λ*i*. *i* < *t* ∧ *ns* ! *i* = *ns* ! *idx*
have *rvt*: *rv t* = *Greatest ?Q*
using *True rv-def* **by** *simp*
moreover have *?Q* (*Greatest ?Q*)
proof (*rule GreatestI-ex-nat*)
show ∃ *k*<*t*. *ns* ! *k* = *ns* ! *idx*
using *True* **by** *simp*
show ⋀*y*. *y* < *t* ∧ *ns* ! *y* = *ns* ! *idx* ⟹ *y* ≤ *t*
by *simp*
qed
ultimately have *?Q* (*rv t*)
by *simp*
then have *rv t* < *t*
by *simp*
then show *?thesis*
by *simp*
next
case *False*
then show *?thesis*
using *rv-def* **by** *auto*
qed

lemma *rv-change*:
assumes *t* < *length ns* **and** *idx* < *length ns* **and** *ns* ! *t* = *ns* ! *idx*
shows *rv* (*Suc t*) = *t*
proof −
let *?P* = λ*i*. *i* < *Suc t* ∧ *ns* ! *i* = *ns* ! *idx*

  have $rv\ (Suc\ t) = Greatest\ ?P$
    using $assms(3)$ $rv\text{-}def$ **by** $auto$
  moreover have $Greatest\ ?P = t$
  proof $(rule\ Greatest\text{-}equality)$
    show $t < Suc\ t \wedge ns\ !\ t = ns\ !\ idx$
      using $assms(3)$ **by** $simp$
    show $\bigwedge y.\ y < Suc\ t \wedge ns\ !\ y = ns\ !\ idx \Longrightarrow y \leq t$
      using $assms$ **by** $simp$
  qed
  ultimately show $?thesis$
    by $simp$
qed

lemma $rv\text{-}keep$:
  assumes $t < length\ ns$ **and** $idx < length\ ns$ **and** $ns\ !\ t \neq ns\ !\ idx$
  shows $rv\ (Suc\ t) = rv\ t$
proof $(cases\ \exists i{<}Suc\ t.\ ns\ !\ i = ns\ !\ idx)$
  case $True$
  let $?Q = \lambda i.\ i < t \wedge ns\ !\ i = ns\ !\ idx$
  have ex: $\exists i{<}t.\ ns\ !\ i = ns\ !\ idx$
    using $True\ assms(3)\ less\text{-}antisym$ **by** $blast$
  then have rvt: $rv\ t = Greatest\ ?Q$
    using $rv\text{-}def$ **by** $simp$
  moreover have $?Q\ (Greatest\ ?Q)$
  proof $(rule\ GreatestI\text{-}ex\text{-}nat)$
    show $\exists k{<}t.\ ns\ !\ k = ns\ !\ idx$
      using ex **.**
    show $\bigwedge y.\ y < t \wedge ns\ !\ y = ns\ !\ idx \Longrightarrow y \leq t$
      by $simp$
  qed
  ultimately have $?Q\ (rv\ t)$
    by $simp$

  let $?P = \lambda i.\ i < Suc\ t \wedge ns\ !\ i = ns\ !\ idx$
  have $rv\ (Suc\ t) = Greatest\ ?P$
    using $True\ rv\text{-}def$ **by** $simp$
  moreover have $Greatest\ ?P = rv\ t$
  proof $(rule\ Greatest\text{-}equality)$
    show $rv\ t < Suc\ t \wedge ns\ !\ rv\ t = ns\ !\ idx$
      using $\langle ?Q\ (rv\ t)\rangle$ **by** $simp$
    show $y \leq rv\ t$ **if** $y < Suc\ t \wedge ns\ !\ y = ns\ !\ idx$ **for** y
    proof $-$
      have $?Q\ y$
        using $True\ assms(3)\ less\text{-}antisym\ that$ **by** $blast$
      moreover have $\forall y.\ ?Q\ y \longrightarrow y \leq t$
        by $simp$
      ultimately have $y \leq Greatest\ ?Q$
        using $Greatest\text{-}le\text{-}nat[of\ ?Q]$ **by** $blast$
      then show $?thesis$
        using rvt **by** $simp$
    qed
  qed
  ultimately show $?thesis$
    by $simp$
next
  case $False$
  then show $?thesis$
    using $rv\text{-}def$ **by** $auto$
qed

definition $tpsL :: nat \Rightarrow tape\ list$ **where**
  $tpsL\ t \equiv tps0$
    $[j2 + 1 := (\lfloor ns\ !\ idx \rfloor_N,\ 1),$

$j2 + 2 := (\lfloor t \rfloor_N, 1)$,
$j2 + 5 := (\lfloor rv\ t \rfloor_N, 1)]$

lemma *tpsL-eq-tps2*: *tpsL 0 = tps2*
using *tpsL-def tps2-def tps0 jk rv-0*
by (*metis add-eq-self-zero add-left-imp-eq gr-implies-not0 less-numeral-extra*(*1*)
list-update-id list-update-swap one-add-one)

definition *tpsC* :: *nat* ⇒ *tape list* **where**
tpsC t ≡ *tps0*
$[j2 + 1 := (\lfloor ns\ !\ idx \rfloor_N, 1)$,
$j2 + 2 := (\lfloor t \rfloor_N, 1)$,
$j2 + 4 := (\lfloor t = idx \rfloor_B, 1)$,
$j2 + 5 := (\lfloor rv\ t \rfloor_N, 1)]$

lemma *tmC*:
assumes *ttt = 3 ∗ nlength* (*min t idx*) + *7*
shows *transforms tmC* (*tpsL t*) *ttt* (*tpsC t*)
unfolding *tmC-def* **by** (*tform tps*: *jk idx tps0 tpsL-def tpsC-def time*: *assms*)

lemma *tmC′* [*transforms-intros*]:
assumes *ttt = 3 ∗ nllength ns* ^ *2* + *7* **and** *t ≤ idx*
shows *transforms tmC* (*tpsL t*) *ttt* (*tpsC t*)
proof −
have *nlength* (*min t idx*) ≤ *nllength ns*
using *idx assms*(*2*) **by** (*metis le-trans length-le-nllength less-or-eq-imp-le min-absorb1 nlength-le-n*)
then have *nlength* (*min t idx*) ≤ *nllength ns* ^ *2*
by (*metis le-square min.absorb2 min.coboundedI1 power2-eq-square*)
then have *3 ∗ nlength* (*min t idx*) + *7* ≤ *3 ∗ nllength ns* ^ *2* + *7*
by *simp*
then show *?thesis*
using *tmC assms*(*1*) *transforms-monotone* **by** *blast*
qed

lemma *condition-tpsC*: (*read* (*tpsC t*)) ! (*j2* + *4*) ≠ □ ⟷ *t = idx*
using *tpsC-def read-ncontents-eq-0 jk* **by** *simp*

definition *tpsB1 t* ≡ *tps0*
$[j2 + 1 := (\lfloor ns\ !\ idx \rfloor_N, 1)$,
$j2 + 2 := (\lfloor t \rfloor_N, 1)$,
$j2 + 3 := (\lfloor ns\ !\ t \rfloor_N, 1)$,
$j2 + 4 := (\lfloor t = idx \rfloor_B, 1)$,
$j2 + 5 := (\lfloor rv\ t \rfloor_N, 1)]$

lemma *tmB1* [*transforms-intros*]:
assumes $ttt = 21 * (nllength\ ns)^2 + 26$ **and** *t < idx*
shows *transforms tmB1* (*tpsC t*) *ttt* (*tpsB1 t*)
unfolding *tmB1-def* **by** (*tform tps*: *jk idx tps0 tpsC-def tpsB1-def time*: *assms idx*)

definition *tpsB2 t* ≡ *tps0*
$[j2 + 1 := (\lfloor ns\ !\ idx \rfloor_N, 1)$,
$j2 + 2 := (\lfloor t \rfloor_N, 1)$,
$j2 + 3 := (\lfloor ns\ !\ t \rfloor_N, 1)$,
$j2 + 4 := (\lfloor ns\ !\ idx = ns\ !\ t \rfloor_B, 1)$,
$j2 + 5 := (\lfloor rv\ t \rfloor_N, 1)]$

lemma *tmB2*:
assumes $ttt = 21 * (nllength\ ns)^2 + 26 + (3 * nlength\ (min\ (ns\ !\ idx)\ (ns\ !\ t)) + 7)$
and *t < idx*
shows *transforms tmB2* (*tpsC t*) *ttt* (*tpsB2 t*)
unfolding *tmB2-def*
proof (*tform tps*: *tpsB1-def jk assms*(*2*) *time*: *assms*(*1*))
show $tpsB2\ t = (tpsB1\ t)[j2 + 4 := (\lfloor ns\ !\ idx = ns\ !\ t \rfloor_B, 1)]$

unfolding *tpsB2-def tpsB1-def* **by** (*simp add*: *list-update-swap*[*of j2* + *4*])
qed

lemma *tmB2'* [*transforms-intros*]:
assumes *ttt* = *24* ∗ (*nllength ns*)^2 + *33* **and** *t* < *idx*
shows *transforms tmB2* (*tpsC t*) *ttt* (*tpsB2 t*)
proof −
let *?ttt* = *21* ∗ (*nllength ns*)^2 + *26* + (*3* ∗ *nlength* (*min* (*ns* ! *idx*) (*ns* ! *t*)) + *7*)
have *?ttt* = *21* ∗ (*nllength ns*)^2 + *33* + *3* ∗ *nlength* (*min* (*ns* ! *idx*) (*ns* ! *t*))
by *simp*
also have ... ≤ *21* ∗ (*nllength ns*)^2 + *33* + *3* ∗ *nllength ns*
using *nlength-min-le-nllength idx assms*(*2*) **by** *simp*
also have ... ≤ *21* ∗ (*nllength ns*)^2 + *33* + *3* ∗ *nllength ns* ^ *2*
by (*simp add*: *power2-eq-square*)
also have ... = *24* ∗ (*nllength ns*)^2 + *33*
by *simp*
finally have *?ttt* ≤ *24* ∗ (*nllength ns*)^2 + *33* **.**
then show *?thesis*
using *assms tmB2 transforms-monotone* **by** *blast*
qed

definition *tpsB3 t* ≡ *tps0*
[*j2* + *1* := (⌊*ns* ! *idx*⌋$_N$, *1*),
j2 + *2* := (⌊*t*⌋$_N$, *1*),
j2 + *4* := (⌊*ns* ! *idx* = *ns* ! *t*⌋$_B$, *1*),
j2 + *5* := (⌊*rv t*⌋$_N$, *1*)]

lemma *condition-tpsB3*: (*read* (*tpsB3 t*)) ! (*j2* + *4*) ≠ □ ⟷ *ns* ! *idx* = *ns* ! *t*
using *tpsB3-def read-ncontents-eq-0 jk* **by** *simp*

lemma *tmB3* [*transforms-intros*]:
assumes *ttt* = *24* ∗ (*nllength ns*)^2 + *33* + (*10* + *2* ∗ *nlength* (*ns* ! *t*)) **and** *t* < *idx*
shows *transforms tmB3* (*tpsC t*) *ttt* (*tpsB3 t*)
unfolding *tmB3-def*
proof (*tform tps*: *assms*(*2*) *tpsB2-def jk time*: *assms*(*1*))
show *tpsB3 t* = (*tpsB2 t*)[*j2* + *3* := (⌊*0*⌋$_N$, *1*)]
(**is** *?l* = *?r*)
proof (*rule nth-equalityI*)
show *length ?l* = *length ?r*
using *tpsB2-def tpsB3-def tps0* **by** *simp*
show *?l* ! *j* = *?r* ! *j* **if** *j* < *length ?l* **for** *j*
proof −
consider *j* = *j1* | *j* = *j2* | *j* = *j2* + *1* | *j* = *j2* + *2* | *j* = *j2* + *3* | *j* = *j2* + *4* | *j* = *j2* + *5*
| *j* ≠ *j1* ∧ *j* ≠ *j2* ∧ *j* ≠ *j2* + *1* ∧ *j* ≠ *j2* + *2* ∧ *j* ≠ *j2* + *3* ∧ *j* ≠ *j2* + *4* ∧ *j* ≠ *j2* + *5*
by *auto*
then show *?thesis*
using *tpsB2-def tpsB3-def tps0 jk*
by (*cases*, *simp-all only*: *nth-list-update-eq nth-list-update-neq length-list-update*, *metis nth-list-update-neq*)
qed
qed
qed

definition *tpsI0 t* ≡ *tps0*
[*j2* + *1* := (⌊*ns* ! *idx*⌋$_N$, *1*),
j2 + *2* := (⌊*t*⌋$_N$, *1*),
j2 + *4* := (⌊*1*⌋$_N$, *1*),
j2 + *5* := (⌊*rv t*⌋$_N$, *1*)]

definition *tpsI1 t* ≡ *tps0*
[*j2* + *1* := (⌊*ns* ! *idx*⌋$_N$, *1*),
j2 + *2* := (⌊*t*⌋$_N$, *1*),
j2 + *5* := (⌊*rv t*⌋$_N$, *1*)]

```
lemma tmI1 [transforms-intros]:
  assumes t < idx and ns ! idx = ns ! t
  shows transforms tmI1 (tpsB3 t) 12 (tpsI1 t)
  unfolding tmI1-def
proof (tform tps: tpsB3-def jk assms(2) time: assms nlength-1-simp)
  show tpsI1 t = (tpsB3 t)[j2 + 4 := (⌊0⌋N, 1)]
    (is ?l = ?r)
  proof (rule nth-equalityI)
    show length ?l = length ?r
      using tpsB3-def tpsI1-def tps0 jk by simp
    show ?l ! j = ?r ! j if j < length ?l for j
    proof −
      consider j = j1 | j = j2 | j = j2 + 1 | j = j2 + 2 | j = j2 + 3 | j = j2 + 4 | j = j2 + 5
        | j ≠ j1 ∧ j ≠ j2 ∧ j ≠ j2 + 1 ∧ j ≠ j2 + 2 ∧ j ≠ j2 + 3 ∧ j ≠ j2 + 4 ∧ j ≠ j2 + 5
        by auto
      then show ?thesis
        using tpsB3-def tpsI1-def tps0 jk
        by (cases, simp-all only: nth-list-update-eq nth-list-update-neq length-list-update, metis nth-list-update-neq)
    qed
  qed
qed

definition tpsI2 t ≡ tps0
    [j2 + 1 := (⌊ns ! idx⌋N, 1),
     j2 + 2 := (⌊t⌋N, 1),
     j2 + 5 := (⌊t⌋N, 1)]

lemma tmI2 [transforms-intros]:
  assumes ttt = 26 + 3 ∗ nlength t + 3 ∗ nlength (rv t)
    and t < idx
    and ns ! idx = ns ! t
  shows transforms tmI2 (tpsB3 t) ttt (tpsI2 t)
  unfolding tmI2-def
proof (tform tps: assms(2,3) tpsI1-def jk time: assms(1))
  show tpsI2 t = (tpsI1 t)[j2 + 5 := (⌊t⌋N, 1)]
    (is ?l = ?r)
  proof (rule nth-equalityI)
    show length ?l = length ?r
      using tpsI1-def tpsI2-def tps0 by simp
    show ?l ! j = ?r ! j if j < length ?l for j
    proof −
      consider j = j1 | j = j2 | j = j2 + 1 | j = j2 + 2 | j = j2 + 3 | j = j2 + 4 | j = j2 + 5
        | j ≠ j1 ∧ j ≠ j2 ∧ j ≠ j2 + 1 ∧ j ≠ j2 + 2 ∧ j ≠ j2 + 3 ∧ j ≠ j2 + 4 ∧ j ≠ j2 + 5
        by auto
      then show ?thesis
        using tpsI1-def tpsI2-def tps0 jk assms(2,3)
      by (cases, simp-all only: One-nat-def nth-list-update-eq nth-list-update-neq length-list-update list-update-overwrite)
    qed
  qed
qed

definition tpsI t ≡ tps0
    [j2 + 1 := (⌊ns ! idx⌋N, 1),
     j2 + 2 := (⌊t⌋N, 1),
     j2 + 5 := (⌊rv (Suc t)⌋N, 1)]

lemma tmI [transforms-intros]:
  assumes ttt = 28 + 6 ∗ nllength ns and t < idx
  shows transforms tmI (tpsB3 t) ttt (tpsI t)
  unfolding tmI-def
proof (tform tps: assms)
  let ?tT = 26 + 3 ∗ nlength t + 3 ∗ nlength (rv t)
  have ∗: (ns ! idx = ns ! t) = (read (tpsB3 t) ! (j2 + 4) ≠ □)
```

```
    using condition-tpsB3 by simp
  then show read (tpsB3 t) ! (j2 + 4) ≠ □ ⟹ ns ! idx = ns ! t
    by simp
  have ns ! idx = ns ! t ⟹ rv (Suc t) = t
    using rv-change idx assms(2) by simp
  then show read (tpsB3 t) ! (j2 + 4) ≠ □ ⟹ tpsI t = tpsI2 t
    using tpsI-def tpsI2-def * by simp

  have ns ! idx ≠ ns ! t ⟹ rv (Suc t) = rv t
    using rv-keep idx assms(2) by simp
  then have ns ! idx ≠ ns ! t ⟹ tpsI t = tpsB3 t
    using tpsI-def tpsB3-def tps0 jk
    by (smt (verit, ccfv-SIG) add-left-imp-eq list-update-id list-update-swap numeral-eq-iff
      one-eq-numeral-iff semiring-norm(83) semiring-norm(87))
  then show ¬ read (tpsB3 t) ! (j2 + 4) ≠ □ ⟹ tpsI t = tpsB3 t
    using * by simp
  show 26 + 3 * nlength t + 3 * nlength (rv t) + 2 ≤ ttt
  proof -
    have 26 + 3 * nlength t + 3 * nlength (rv t) + 2 = 28 + 3 * nlength t + 3 * nlength (rv t)
      by simp
    also have ... ≤ 28 + 3 * nlength idx + 3 * nlength (rv t)
      using assms(2) nlength-mono by simp
    also have ... ≤ 28 + 3 * nlength idx + 3 * nlength idx
    proof -
      have rv t ≤ idx
        using assms(2) rv-le by (metis less-or-eq-imp-le max-absorb2)
      then show ?thesis
        using nlength-mono by simp
    qed
    also have ... = 28 + 6 * nlength idx
      by simp
    also have ... ≤ 28 + 6 * nllength ns
    proof -
      have nlength idx ≤ nlength (length ns)
        using idx nlength-mono by simp
      then have nlength idx ≤ length ns
        using nlength-le-n by (meson le-trans)
      then have nlength idx ≤ nllength ns
        using length-le-nllength le-trans by blast
      then show ?thesis
        by simp
    qed
    finally show ?thesis
      using assms(1) by simp
  qed
qed

lemma tmB4:
  assumes ttt = 71 + 24 * (nllength ns)^2 + 2 * nlength (ns ! t) + 6 * nllength ns
    and t < idx
  shows transforms tmB4 (tpsC t) ttt (tpsI t)
  unfolding tmB4-def by (tform tps: assms(2) jk time: assms(1))

lemma tmB4' [transforms-intros]:
  assumes ttt = 71 + 32 * (nllength ns)^2 and t < idx
  shows transforms tmB4 (tpsC t) ttt (tpsI t)
proof -
  let ?ttt = 71 + 24 * (nllength ns)^2 + 2 * nlength (ns ! t) + 6 * nllength ns
  have ?ttt ≤ 71 + 24 * (nllength ns)^2 + 2 * nlength (ns ! t) + 6 * nllength ns ^ 2
    by (metis add-mono-thms-linordered-semiring(2) le-square mult.commute mult-le-mono1 power2-eq-square)
  also have ... = 71 + 30 * (nllength ns)^2 + 2 * nlength (ns ! t)
    by simp
  also have ... ≤ 71 + 30 * (nllength ns)^2 + 2 * nllength ns
```

using *member-le-nllength assms*(*2*) *idx* **by** *simp*
also have ... $\leq$ *71* + *30* $*$ (*nllength ns*)^2 + *2* $*$ *nllength ns* ^ *2*
by (*simp add*: *power2-eq-square*)
also have ... = *71* + *32* $*$ (*nllength ns*)^2
by *simp*
finally have *?ttt* $\leq$ *71* + *32* $*$ (*nllength ns*)^2 **.**
then show *?thesis*
using *assms tmB4 transforms-monotone* **by** *blast*
qed

definition *tpsB5 t* $\equiv$ *tps0*
[*j2* + *1* := ($\lfloor$*ns* ! *idx*$\rfloor_N$, *1*),
j2 + *2* := ($\lfloor$*Suc t*$\rfloor_N$, *1*),
j2 + *5* := ($\lfloor$*rv* (*Suc t*)$\rfloor_N$, *1*)]

lemma *tmB5*:
assumes *ttt* = *76* + *32* $*$ (*nllength ns*)^2 + *2* $*$ *nlength t* **and** *t* < *idx*
shows *transforms tmB5* (*tpsC t*) *ttt* (*tpsB5 t*)
unfolding *tmB5-def*
proof (*tform tps*: *assms*(*2*) *tpsI-def jk time*: *assms*(*1*))
show *tpsB5 t* = (*tpsI t*)[*j2* + *2* := ($\lfloor$*Suc t*$\rfloor_N$, *1*)]
using *tpsB5-def tpsI-def*
by (*smt* (*verit*) *add-left-imp-eq list-update-overwrite list-update-swap numeral-eq-iff semiring-norm*(*89*))
qed

lemma *tmB5′* [*transforms-intros*]:
assumes *ttt* = *76* + *34* $*$ (*nllength ns*)^2 **and** *t* < *idx*
shows *transforms tmB5* (*tpsC t*) *ttt* (*tpsL* (*Suc t*))
proof −
have *76* + *32* $*$ (*nllength ns*)^2 + *2* $*$ *nlength t* $\leq$ *76* + *32* $*$ (*nllength ns*)^2 + *2* $*$ *nllength ns*
using *assms*(*2*) *idx length-le-nllength nlength-le-n*
by (*meson add-mono-thms-linordered-semiring*(*2*) *le-trans less-or-eq-imp-le mult-le-mono2*)
also have ... $\leq$ *76* + *32* $*$ (*nllength ns*)^2 + *2* $*$ *nllength ns* ^ *2*
by (*simp add*: *power2-eq-square*)
also have ... $\leq$ *76* + *34* $*$ (*nllength ns*)^2
by *simp*
finally have *76* + *32* $*$ (*nllength ns*)^2 + *2* $*$ *nlength t* $\leq$ *76* + *34* $*$ (*nllength ns*)^2 **.**
moreover have *tpsL* (*Suc t*) = *tpsB5 t*
using *tpsL-def tpsB5-def* **by** *simp*
ultimately show *?thesis*
using *assms tmB5 transforms-monotone* **by** *fastforce*
qed

lemma *tmL* [*transforms-intros*]:
assumes *ttt* = *125* $*$ *nllength ns* ^ *3* + *8* **and** *iidx* = *idx*
shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsC iidx*)
unfolding *tmL-def*
proof (*tform tps*: *assms*)
let *?tC* = *3* $*$ *nllength ns* ^ *2* + *7*
let *?tB5* = *76* + *34* $*$ (*nllength ns*)^2
show $\bigwedge$*t*. *t* < *iidx* $\Longrightarrow$ *read* (*tpsC t*) ! (*j2* + *4*) = $\Box$
using *condition-tpsC assms*(*2*) **by** *fast*
show *read* (*tpsC iidx*) ! (*j2* + *4*) $\neq$ $\Box$
using *condition-tpsC assms*(*2*) **by** *simp*
have *iidx* $*$ (*?tC* + *?tB5* + *2*) + *?tC* + *1* = *iidx* $*$ (*37* $*$ (*nllength ns*)^2 + *85*) + *3* $*$ (*nllength ns*)^2 + *8*
by *simp*
also have ... $\leq$ *length ns* $*$ (*37* $*$ (*nllength ns*)^2 + *85*) + *3* $*$ (*nllength ns*)^2 + *8*
using *assms*(*2*) *idx* **by** *simp*
also have ... $\leq$ *nllength ns* $*$ (*37* $*$ (*nllength ns*)^2 + *85*) + *3* $*$ (*nllength ns*)^2 + *8*
using *length-le-nllength* **by** *simp*
also have ... = *37* $*$ *nllength ns* ^ *3* + *85* $*$ *nllength ns* + *3* $*$ (*nllength ns*)^2 + *8*
by *algebra*
also have ... $\leq$ *37* $*$ *nllength ns* ^ *3* + *85* $*$ *nllength ns* + *3* $*$ *nllength ns* ^ *3* + *8*

proof −
have *nllength ns* ^ *2* ≤ *nllength ns* ^ *3*
by (*metis Suc-eq-plus1 add.commute eq-refl le-add2 neq0-conv numeral-3-eq-3 numerals*(*2*)
pow-mono power-eq-0-iff zero-less-Suc)
then show *?thesis*
by *simp*
qed
also have ... ≤ *37* ∗ *nllength ns* ^ *3* + *85* ∗ *nllength ns* ^ *3* + *3* ∗ *nllength ns* ^ *3* + *8*
by (*simp add*: *power3-eq-cube*)
also have ... = *125* ∗ *nllength ns* ^ *3* + *8*
by *simp*
finally have *iidx* ∗ (*?tC* + *?tB5* + *2*) + *?tC* + *1* ≤ *125* ∗ *nllength ns* ^ *3* + *8* **.**
then show *iidx* ∗ (*?tC* + *?tB5* + *2*) + *?tC* + *1* ≤ *ttt*
using *assms*(*1*) **by** *simp*
qed

lemma *tm2′* [*transforms-intros*]:
assumes *ttt* = *14* + *3* ∗ *nlength idx* + (*21* ∗ (*nllength ns*)^2 + *26*)
shows *transforms tm2 tps0 ttt* (*tpsL 0*)
using *tm2 assms tpsL-eq-tps2* **by** *simp*

lemma *tm3* [*transforms-intros*]:
assumes *ttt* = *40* + (*3* ∗ *nlength idx* + *21* ∗ (*nllength ns*)^2) + (*125* ∗ *nllength ns* ^ *3* + *8*)
shows *transforms tm3 tps0 ttt* (*tpsC idx*)
unfolding *tm3-def* **by** (*tform tps*: *assms jk*)

lemma *tpsC-idx*:
tpsC idx = *tps0*
[*j2* + *1* := (⌊*ns* ! *idx*⌋$_N$, *1*),
j2 + *2* := (⌊*idx*⌋$_N$, *1*),
j2 + *4* := (⌊*1*⌋$_N$, *1*),
j2 + *5* := (⌊*if* ∃ *i*<*idx*. *ns* ! *i* = *ns* ! *idx then GREATEST i*. *i* < *idx* ∧ *ns* ! *i* = *ns* ! *idx else idx*⌋$_N$, *1*)]
using *tpsC-def rv-def* **by** *simp*

definition *tps4* :: *tape list* **where**
tps4 ≡ *tps0*
[*j2* + *1* := (⌊[]⌋, *1*),
j2 + *2* := (⌊*idx*⌋$_N$, *1*),
j2 + *4* := (⌊*1*⌋$_N$, *1*),
j2 + *5* := (⌊*if* ∃ *i*<*idx*. *ns* ! *i* = *ns* ! *idx then GREATEST i*. *i* < *idx* ∧ *ns* ! *i* = *ns* ! *idx else idx*⌋$_N$, *1*)]

lemma *tm4* [*transforms-intros*]:
assumes *ttt* = *55* + *3* ∗ *nlength idx* + *21* ∗ (*nllength ns*)^2 + *125* ∗ *nllength ns* ^ *3* + *2* ∗ *nlength* (*ns* ! *idx*)
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform tps*: *assms tpsC-def tps4-def jk*)
show *proper-symbols* (*canrepr* (*ns* ! *idx*))
using *proper-symbols-canrepr* **by** *simp*
show *tps4* = (*tpsC idx*)[*j2* + *1* := (⌊[]⌋, *1*)]
using *tpsC-idx tps4-def* **by** (*simp add*: *list-update-swap*[*of Suc j2*])
qed

definition *tps5* :: *tape list* **where**
tps5 ≡ *tps0*
[*j2* + *1* := (⌊[]⌋, *1*),
j2 + *2* := (⌊[]⌋, *1*),
j2 + *4* := (⌊*1*⌋$_N$, *1*),
j2 + *5* := (⌊*if* ∃ *i*<*idx*. *ns* ! *i* = *ns* ! *idx then GREATEST i*. *i* < *idx* ∧ *ns* ! *i* = *ns* ! *idx else idx*⌋$_N$, *1*)]

lemma *tm5* [*transforms-intros*]:
assumes *ttt* = *62* + *5* ∗ *nlength idx* + *21* ∗ (*nllength ns*)^2 + *125* ∗ *nllength ns* ^ *3* + *2* ∗ *nlength* (*ns* ! *idx*)
shows *transforms tm5 tps0 ttt tps5*
unfolding *tm5-def*

proof (*tform tps*: *tps4-def tps5-def jk time*: *assms tps4-def jk*)
  show *proper-symbols* (*canrepr idx*)
    using *proper-symbols-canrepr* **by** *simp*
qed

definition *tps6* :: *tape list* **where**
  $tps6 \equiv tps0$
    $[j2 + 1 := (\lfloor [] \rfloor, 1),$
     $j2 + 2 := (\lfloor [] \rfloor, 1),$
     $j2 + 4 := (\lfloor [] \rfloor, 1),$
     $j2 + 5 := (\lfloor if\ \exists i<idx.\ ns\ !\ i = ns\ !\ idx\ then\ GREATEST\ i.\ i < idx \wedge ns\ !\ i = ns\ !\ idx\ else\ idx \rfloor_N, 1)]$

lemma *tm6*:
  assumes $ttt = 71 + 5 * nlength\ idx + 21 * (nllength\ ns)^2 + 125 * nllength\ ns\ \hat{}\ 3 + 2 * nlength\ (ns\ !\ idx)$
  shows *transforms tm6 tps0 ttt tps6*
  unfolding *tm6-def*
proof (*tform tps*: *tps5-def tps6-def jk time*: *assms tps5-def jk nlength-1-simp*)
  show *proper-symbols* (*canrepr* (*Suc 0*))
    using *proper-symbols-canrepr* **by** *simp*
qed

definition *tps6'* :: *tape list* **where**
  $tps6' \equiv tps0$
    $[j2 + 5 := (\lfloor if\ \exists i<idx.\ ns\ !\ i = ns\ !\ idx\ then\ GREATEST\ i.\ i < idx \wedge ns\ !\ i = ns\ !\ idx\ else\ idx \rfloor_N, 1)]$

lemma *tps6'-eq-tps6*: $tps6' = tps6$
  using *tps6'-def tps6-def tps0 jk canrepr-0* **by** (*metis* (*no-types*, *lifting*) *list-update-id*)

lemma *tm6'*:
  assumes $ttt = 71 + 153 * nllength\ ns\ \hat{}\ 3$
  shows *transforms tm6 tps0 ttt tps6'*
proof −
  let $?ttt = 71 + 5 * nlength\ idx + 21 * (nllength\ ns)^2 + 125 * nllength\ ns\ \hat{}\ 3 + 2 * nlength\ (ns\ !\ idx)$
  have $?ttt \le 71 + 5 * nlength\ idx + 21 * (nllength\ ns)\hat{}3 + 125 * nllength\ ns\ \hat{}\ 3 + 2 * nlength\ (ns\ !\ idx)$
    using *pow-mono*[*of 2 3 nllength ns*] **by** *fastforce*
  also have ... $= 71 + 5 * nlength\ idx + 146 * nllength\ ns\ \hat{}\ 3 + 2 * nlength\ (ns\ !\ idx)$
    by *simp*
  also have ... $\le 71 + 5 * nllength\ ns + 146 * nllength\ ns\ \hat{}\ 3 + 2 * nlength\ (ns\ !\ idx)$
  proof −
    have $nlength\ idx \le nllength\ ns$
      using *idx* **by** (*meson le-trans length-le-nllength nlength-le-n order.strict-implies-order*)
    then show *?thesis*
      by *simp*
  qed
  also have ... $\le 71 + 5 * nllength\ ns\ \hat{}\ 3 + 146 * nllength\ ns\ \hat{}\ 3 + 2 * nlength\ (ns\ !\ idx)$
    using *linear-le-pow* **by** *simp*
  also have ... $= 71 + 151 * nllength\ ns\ \hat{}\ 3 + 2 * nlength\ (ns\ !\ idx)$
    by *simp*
  also have ... $\le 71 + 151 * nllength\ ns\ \hat{}\ 3 + 2 * nllength\ ns$
    using *idx member-le-nllength* **by** *simp*
  also have ... $\le 71 + 151 * nllength\ ns\ \hat{}\ 3 + 2 * nllength\ ns\hat{}3$
    using *linear-le-pow* **by** *simp*
  also have ... $= 71 + 153 * nllength\ ns\ \hat{}\ 3$
    by *simp*
  finally have $?ttt \le ttt$
    using *assms* **by** *simp*
  then have *transforms tm6 tps0 ttt tps6*
    using *tm6 transforms-monotone* **by** *simp*
  then show *?thesis*
    using *tps6'-eq-tps6* **by** *simp*
qed

end

end

lemma *transforms-tm-prevI* [*transforms-intros*]:
  fixes *j1 j2* :: *tapeidx*
  fixes *tps tps'* :: *tape list* **and** *k idx* :: *nat* **and** *ns* :: *nat list*
  assumes $0 < j1$ $j1 < j2$ $j2 + 6 \leq k$ *length tps* $= k$
    and *idx* $<$ *length ns*
  assumes
    tps ! *j1* $= (\lfloor ns \rfloor_{NL}, 1)$
    tps ! *j2* $= (\lfloor idx \rfloor_N, 1)$
    tps ! $(j2 + 1) = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j2 + 2) = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j2 + 3) = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j2 + 4) = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j2 + 5) = (\lfloor 0 \rfloor_N, 1)$
  assumes *ttt* $= 71 + 153 *$ *nllength ns* ^ *3*
  assumes *tps'* $=$ *tps*
    $[j2 + 5 := (\lfloor$*previous ns idx*$\rfloor_N, 1)]$
  shows *transforms* (*tm-prev j1 j2*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-prev j1 j2* **.**
  show *?thesis*
    using *assms loc.tm6-eq-tm-prev loc.tm6' loc.tps6'-def previous-def* **by** *simp*
qed

2.8.7 Checking containment in a list

A Turing machine that checks whether a number given on tape j_2 is contained in the list of numbers on tape j_1. If so, it writes a 1 to tape j_3, and otherwise leaves tape j_3 unchanged. So tape j_3 should be initialized with 0.

definition *tm-contains* :: *tapeidx* ⇒ *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-contains j1 j2 j3 ≡
    WHILE [] ; λ*rs. rs* ! *j1* ≠ □ *DO*
      tm-nextract | *j1* (*j3* + *1*) ;;
      tm-equalsn j2 (*j3* + *1*) (*j3* + *2*) ;;
      IF λ*rs. rs* ! (*j3* + *2*) ≠ □ *THEN*
        tm-setn j3 1
      ELSE
        []
      ENDIF ;;
      tm-setn (*j3* + *1*) *0* ;;
      tm-setn (*j3* + *2*) *0*
    DONE ;;
    tm-cr j1

lemma *tm-contains-tm*:
  assumes $0 < j1$ $j1 \neq j2$ $j3 + 2 < k$ $j1 < j3$ $j2 < j3$ **and** $k \geq 2$ **and** $G \geq 5$
  shows *turing-machine k G* (*tm-contains j1 j2 j3*)
  unfolding *tm-contains-def*
  using *tm-nextract-tm tm-equalsn-tm Nil-tm tm-setn-tm tm-cr-tm assms*
    turing-machine-branch-turing-machine turing-machine-loop-turing-machine
  by *simp*

locale *turing-machine-contains* =
  fixes *j1 j2 j3* :: *tapeidx*
begin

definition *tmL1* ≡ *tm-nextract* | *j1* (*j3* + *1*)
definition *tmL2* ≡ *tmL1* ;; *tm-equalsn j2* (*j3* + *1*) (*j3* + *2*)
definition *tmI* ≡ *IF* λ*rs. rs* ! (*j3* + *2*) ≠ □ *THEN tm-setn j3 1 ELSE* [] *ENDIF*
definition *tmL3* ≡ *tmL2* ;; *tmI*
definition *tmL4* ≡ *tmL3* ;; *tm-setn* (*j3* + *1*) *0*

definition *tmL5* ≡ *tmL4* ;; *tm-setn* (*j3* + *2*) *0*
definition *tmL* ≡ *WHILE* [] ; *λrs. rs* ! *j1* ≠ □ *DO tmL5 DONE*
definition *tm2* ≡ *tmL* ;; *tm-cr j1*

lemma *tm2-eq-tm-contains*: *tm2* = *tm-contains j1 j2 j3*
  unfolding *tm2-def tmL-def tmL5-def tmL4-def tmL3-def tmI-def tmL2-def tmL1-def tm-contains-def*
  by *simp*

context
  fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *ns* :: *nat list* **and** *needle* :: *nat*
  assumes *jk*: *0* < *j1 j1* ≠ *j2 j3* + *2* < *k j1* < *j3 j2* < *j3 length tps0* = *k*
    and *tps0*:
      tps0 ! *j1* = *nltape′ ns 0*
      tps0 ! *j2* = (⌊*needle*⌋$_N$, *1*)
      tps0 ! *j3* = (⌊*0*⌋$_N$, *1*)
      tps0 ! (*j3* + *1*) = (⌊*0*⌋$_N$, *1*)
      tps0 ! (*j3* + *2*) = (⌊*0*⌋$_N$, *1*)
begin

definition *tpsL* :: *nat* ⇒ *tape list* **where**
  tpsL t ≡ *tps0*
    [*j1* := *nltape′ ns t*,
     j3 := (⌊∃ *i*<*t*. *ns* ! *i* = *needle*⌋$_B$, *1*)]

lemma *tpsL0*: *tpsL 0* = *tps0*
  unfolding *tpsL-def* **using** *tps0* **by** (*smt* (*verit*) *list-update-id not-less-zero*)

definition *tpsL1* :: *nat* ⇒ *tape list* **where**
  tpsL1 t ≡ *tps0*
    [*j1* := *nltape′ ns* (*Suc t*),
     j3 := (⌊∃ *i*<*t*. *ns* ! *i* = *needle*⌋$_B$, *1*),
     j3 + *1* := (⌊*ns* ! *t*⌋$_N$, *1*)]

lemma *tmL1* [*transforms-intros*]:
  assumes *ttt* = *12* + *2* ∗ *nlength* (*ns* ! *t*) **and** *t* < *length ns*
  shows *transforms tmL1* (*tpsL t*) *ttt* (*tpsL1 t*)
  unfolding *tmL1-def*
proof (*tform tps*: *assms*(*2*) *tpsL-def tpsL1-def jk tps0*)
  show *ttt* = *12* + *2* ∗ *nlength 0* + *2* ∗ *nlength* (*ns* ! *t*)
    using *assms*(*1*) **by** *simp*
  show *tpsL1 t* = (*tpsL t*)
    [*j1* := *nltape′ ns* (*Suc t*),
     j3 + *1* := (⌊*ns* ! *t*⌋$_N$, *1*)]
    unfolding *tpsL1-def tpsL-def* **using** *jk* **by** (*simp add*: *list-update-swap*)
qed

definition *tpsL2* :: *nat* ⇒ *tape list* **where**
  tpsL2 t ≡ *tps0*
    [*j1* := *nltape′ ns* (*Suc t*),
     j3 := (⌊∃ *i*<*t*. *ns* ! *i* = *needle*⌋$_B$, *1*),
     j3 + *1* := (⌊*ns* ! *t*⌋$_N$, *1*),
     j3 + *2* := (⌊*needle* = *ns* ! *t*⌋$_B$, *1*)]

lemma *tmL2* [*transforms-intros*]:
  assumes *ttt* = *12* + *2* ∗ *nlength* (*ns* ! *t*) + (*3* ∗ *nlength* (*min needle* (*ns* ! *t*)) + *7*)
    and *t* < *length ns*
  shows *transforms tmL2* (*tpsL t*) *ttt* (*tpsL2 t*)
  unfolding *tmL2-def* **by** (*tform tps*: *assms tps0 tpsL1-def tpsL2-def jk*)

definition *tpsI* :: *nat* ⇒ *tape list* **where**
  tpsI t ≡ *tps0*
    [*j1* := *nltape′ ns* (*Suc t*),
     j3 := (⌊∃ *i*<*Suc t*. *ns* ! *i* = *needle*⌋$_B$, *1*),

j3 + 1 := (⌊ns ! t⌋$_N$, 1),
j3 + 2 := (⌊needle = ns ! t⌋$_B$, 1)]

lemma *tmI* [*transforms-intros*]:
assumes *ttt = 16* **and** *t < length ns*
shows *transforms tmI* (*tpsL2 t*) *ttt* (*tpsI t*)
unfolding *tmI-def*
proof (*tform tps*: *tpsL2-def jk*)
show *10 + 2 ∗ nlength* (*if ∃ i<t. ns ! i = needle then 1 else 0*) *+ 2 ∗ nlength 1 + 2 ≤ ttt*
using *assms*(*1*) *nlength-1-simp nlength-0* **by** *simp*
show *0 + 1 ≤ ttt*
using *assms*(*1*) **by** *simp*

have *tpsL2 t !* (*j3 + 2*) = (⌊*needle = ns ! t*⌋$_B$, *1*)
using *tpsL2-def jk* **by** *simp*
then have ∗: (*read* (*tpsL2 t*) *!* (*j3 + 2*) = □) = (*needle ≠ ns ! t*)
using *jk read-ncontents-eq-0*[*of tpsL2 t j3 + 2*] *tpsL2-def* **by** *simp*

show *tpsI t =* (*tpsL2 t*)[*j3 :=* (⌊*1*⌋$_N$, *1*)] **if** *read* (*tpsL2 t*) *!* (*j3 + 2*) ≠ □
proof −
have *needle = ns ! t*
using ∗ *that* **by** *simp*
then have ∃ *i<Suc t. ns ! i = needle*
using *assms*(*2*) **by** *auto*
then show *?thesis*
unfolding *tpsI-def tpsL2-def* **using** *jk* **by** (*simp add*: *list-update-swap*)
qed
show *tpsI t = tpsL2 t* **if** ¬ *read* (*tpsL2 t*) *!* (*j3 + 2*) ≠ □
proof −
have *needle ≠ ns ! t*
using ∗ *that* **by** *simp*
then have (∃ *i<Suc t. ns ! i = needle*) = (∃ *i< t. ns ! i = needle*)
using *assms*(*2*) *less-Suc-eq* **by** *auto*
then show *?thesis*
unfolding *tpsI-def tpsL2-def* **by** *simp*
qed
qed

lemma *tmL3* [*transforms-intros*]:
assumes *ttt = 28 + 2 ∗ nlength* (*ns ! t*) + (*3 ∗ nlength* (*min needle* (*ns ! t*)) *+ 7*)
and *t < length ns*
shows *transforms tmL3* (*tpsL t*) *ttt* (*tpsI t*)
unfolding *tmL3-def* **by** (*tform tps*: *assms*)
definition *tpsL4* :: *nat ⇒ tape list* **where**
tpsL4 t ≡ tps0
[*j1 := nltape′ ns* (*Suc t*),
j3 := (⌊∃ *i<Suc t. ns ! i = needle*⌋$_B$, *1*),
j3 + 1 := (⌊*0*⌋$_N$, *1*),
j3 + 2 := (⌊*needle = ns ! t*⌋$_B$, *1*)]

lemma *tmL4* [*transforms-intros*]:
assumes *ttt = 38 + 4 ∗ nlength* (*ns ! t*) + (*3 ∗ nlength* (*min needle* (*ns ! t*)) *+ 7*)
and *t < length ns*
shows *transforms tmL4* (*tpsL t*) *ttt* (*tpsL4 t*)
unfolding *tmL4-def*
proof (*tform tps*: *assms tpsI-def jk*)
show *tpsL4 t =* (*tpsI t*)[*j3 + 1 :=* (⌊*0*⌋$_N$, *1*)]
unfolding *tpsL4-def tpsI-def* **by** (*simp add*: *list-update-swap*)
qed

definition *tpsL5* :: *nat ⇒ tape list* **where**
tpsL5 t ≡ tps0
[*j1 := nltape′ ns* (*Suc t*),

$j3 := (\lfloor\exists\, i<Suc\ t.\ ns\ !\ i = needle\rfloor_B,\ 1),$
$j3 + 1 := (\lfloor 0\rfloor_N,\ 1),$
$j3 + 2 := (\lfloor 0\rfloor_N,\ 1)]$

lemma *tmL5*:
assumes $ttt = 48 + 4 * nlength\ (ns\ !\ t) + (3 * nlength\ (min\ needle\ (ns\ !\ t)) + 7) +$
$2 * nlength\ (if\ needle = ns\ !\ t\ then\ 1\ else\ 0)$
and $t < length\ ns$
shows *transforms tmL5* (*tpsL t*) *ttt* (*tpsL5 t*)
unfolding *tmL5-def*
proof (*tform tps*: *assms tpsL4-def jk*)
show $tpsL5\ t = (tpsL4\ t)[j3 + 2 := (\lfloor 0\rfloor_N,\ 1)]$
unfolding *tpsL5-def tpsL4-def* **by** (*simp add*: *list-update-swap*)
qed

definition *tpsL5′* :: *nat* ⇒ *tape list* **where**
$tpsL5'\ t \equiv tps0$
$[j1 := nltape'\ ns\ (Suc\ t),$
$j3 := (\lfloor\exists\, i<Suc\ t.\ ns\ !\ i = needle\rfloor_B,\ 1)]$

lemma *tpsL5′*: $tpsL5'\ t = tpsL5\ t$
using *tpsL5′-def tpsL5-def tps0 jk*
by (*smt* (*verit*, *del-insts*) *One-nat-def less-Suc-eq less-add-same-cancel1 list-update-swap not-less-eq tape-list-eq zero-less-numeral*)

lemma *tmL5′*:
assumes $ttt = 57 + 4 * nlength\ (ns\ !\ t) + 3 * nlength\ (min\ needle\ (ns\ !\ t))$
and $t < length\ ns$
shows *transforms tmL5* (*tpsL t*) *ttt* (*tpsL5′ t*)
proof −
have $nlength\ (if\ needle = ns\ !\ t\ then\ 1\ else\ 0) \le 1$
using *nlength-1-simp* **by** *simp*
then have $48 + 4 * nlength\ (ns\ !\ t) + (3 * nlength\ (min\ needle\ (ns\ !\ t)) + 7) +$
$2 * nlength\ (if\ needle = ns\ !\ t\ then\ 1\ else\ 0) \le ttt$
using *assms*(*1*) **by** *simp*
then show *?thesis*
using *tpsL5′ tmL5 transforms-monotone assms*(*2*) **by** *fastforce*
qed

lemma *tmL5′′* [*transforms-intros*]:
assumes $ttt = 57 + 7 * nllength\ ns$ **and** $t < length\ ns$
shows *transforms tmL5* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
proof −
have $nlength\ (ns\ !\ t) \le nllength\ ns$
using *assms*(*2*) **by** (*simp add*: *member-le-nllength*)
moreover from *this* **have** $nlength\ (min\ needle\ (ns\ !\ t)) \le nllength\ ns$
using *nlength-mono* **by** (*metis dual-order.trans min-def*)
ultimately have $ttt \ge 57 + 4 * nlength\ (ns\ !\ t) + 3 * nlength\ (min\ needle\ (ns\ !\ t))$
using *assms*(*1*) **by** *simp*
moreover have $tpsL5'\ t = tpsL\ (Suc\ t)$
using *tpsL5′-def tpsL-def* **by** *simp*
ultimately show *?thesis*
using *tmL5′ assms*(*2*) *transforms-monotone* **by** *fastforce*
qed

lemma *tmL* [*transforms-intros*]:
assumes $ttt = length\ ns * (59 + 7 * nllength\ ns) + 1$
shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsL* (*length ns*))
unfolding *tmL-def*
proof (*tform*)
let $?t = 57 + 7 * nllength\ ns$
show $length\ ns * (57 + 7 * nllength\ ns + 2) + 1 \le ttt$
using *assms* **by** *simp*

```
  have *: tpsL t ! j1 = nltape' ns t for t
    using tpsL-def jk by simp
  moreover have read (tpsL t) ! j1 = tpsL t :.: j1 for t
    using tapes-at-read'[of j1 tpsL t] tpsL-def jk by simp
  ultimately have read (tpsL t) ! j1 = |.| (nltape' ns t) for t
    by simp
  then have read (tpsL t) ! j1 = □ ⟷ (t ≥ length ns) for t
    using nltape'-tape-read by simp
  then show
      ⋀i. i < length ns ⟹ read (tpsL i) ! j1 ≠ □
      ¬ read (tpsL (length ns)) ! j1 ≠ □
    using * by simp-all
qed

definition tps2 :: tape list where
  tps2 ≡ tps0
    [j1 := nltape' ns 0,
     j3 := (⌊∃ i<length ns. ns ! i = needle⌋_B, 1)]

lemma tm2:
  assumes ttt = length ns * (59 + 7 * nllength ns) + nllength ns + 4
  shows transforms tm2 (tpsL 0) ttt tps2
  unfolding tm2-def
proof (tform tps: tpsL-def jk)
  show clean-tape (tpsL (length ns) ! j1)
    using tpsL-def jk clean-tape-nlcontents by simp
  have tpsL (length ns) ! j1 |#=| 1 = nltape' ns 0
    using tpsL-def jk by simp
  then have (tpsL (length ns))[j1 := tpsL (length ns) ! j1 |#=| 1] = (tpsL (length ns))[j1 := nltape' ns 0]
    by simp
  then show tps2 = (tpsL (length ns))[j1 := tpsL (length ns) ! j1 |#=| 1]
    unfolding tps2-def tpsL-def using jk by (simp add: list-update-swap)
  have tpsL (length ns) :#: j1 = Suc (nllength ns)
    using tpsL-def jk by simp
  then show ttt = length ns * (59 + 7 * nllength ns) + 1 +
      (tpsL (length ns) :#: j1 + 2)
    using assms by simp
qed

definition tps2' :: tape list where
  tps2' ≡ tps0
    [j3 := (⌊needle ∈ set ns⌋_B, 1)]

lemma tm2':
  assumes ttt = 67 * nllength ns ^ 2 + 4
  shows transforms tm2 tps0 ttt tps2'
proof -
  let ?t = length ns * (59 + 7 * nllength ns) + nllength ns + 4
  have ?t ≤ nllength ns * (59 + 7 * nllength ns) + nllength ns + 4
    by (simp add: length-le-nllength)
  also have ... = 60 * nllength ns + 7 * nllength ns ^ 2 + 4
    by algebra
  also have ... ≤ 60 * nllength ns ^ 2 + 7 * nllength ns ^ 2 + 4
    using linear-le-pow by simp
  also have ... = 67 * nllength ns ^ 2 + 4
    by simp
  finally have ?t ≤ 67 * nllength ns ^ 2 + 4 .
  moreover have tps2' = tps2
    unfolding tps2-def tps2'-def using tps0(1) by (smt (verit) in-set-conv-nth list-update-id)
  ultimately show ?thesis
    using tm2 assms transforms-monotone tpsL0 by simp
qed
```

end

end

lemma *transforms-tm-containsI* [*transforms-intros*]:
  fixes *j1 j2 j3* :: *tapeidx*
  fixes *tps tps'* :: *tape list* **and** *ttt k needle* :: *nat* **and** *ns* :: *nat list*
  assumes $0 < j1$ $j1 \neq j2$ $j3 + 2 < k$ $j1 < j3$ $j2 < j3$ *length tps* $= k$
  assumes
    tps ! *j1* = *nltape' ns 0*
    tps ! *j2* = $(\lfloor needle \rfloor_N, 1)$
    tps ! *j3* = $(\lfloor 0 \rfloor_N, 1)$
    tps ! $(j3 + 1) = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j3 + 2) = (\lfloor 0 \rfloor_N, 1)$
  assumes *ttt* = 67 ∗ *nllength ns* ^ 2 + 4
  assumes *tps'* = *tps*
    [*j3* := $(\lfloor needle \in set\ ns \rfloor_B, 1)$]
  shows *transforms* (*tm-contains j1 j2 j3*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-contains j1 j2 j3* **.**
  show *?thesis*
    using *assms loc.tm2-eq-tm-contains loc.tps2'-def loc.tm2'* **by** *simp*
qed

2.8.8 Creating lists of consecutive numbers

The next TM accepts a number *start* on tape j_1 and a number *delta* on tape j_2. It outputs the list $[start, \ldots, start + delta - 1]$ on tape j_3.

definition *tm-range* :: *tapeidx* ⇒ *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-range j1 j2 j3 ≡
    tm-copyn j1 (*j3* + 2) ;;
    tm-copyn j2 (*j3* + 1) ;;
    WHILE [] ; λ*rs. rs* ! (*j3* + 1) ≠ □ *DO*
      tm-append j3 (*j3* + 2) ;;
      tm-incr (*j3* + 2) ;;
      tm-decr (*j3* + 1)
    DONE ;;
    tm-setn (*j3* + 2) *0* ;;
    tm-cr j3

lemma *tm-range-tm*:
  assumes $k \geq j3 + 3$ **and** $G \geq 5$ **and** $j1 \neq j2$ **and** $0 < j1$ **and** $0 < j2$ **and** $j1 < j3$ **and** $j2 < j3$
  shows *turing-machine k G* (*tm-range j1 j2 j3*)
  unfolding *tm-range-def*
  using *assms tm-copyn-tm tm-decr-tm tm-append-tm tm-setn-tm tm-incr-tm Nil-tm tm-cr-tm*
    turing-machine-loop-turing-machine
  by *simp*

locale *turing-machine-range* =
  fixes *j1 j2 j3* :: *tapeidx*
begin

definition *tm1* ≡ *tm-copyn j1* (*j3* + 2)
definition *tm2* ≡ *tm1* ;; *tm-copyn j2* (*j3* + 1)
definition *tmB1* ≡ *tm-append j3* (*j3* + 2)
definition *tmB2* ≡ *tmB1* ;; *tm-incr* (*j3* + 2)
definition *tmB3* ≡ *tmB2* ;; *tm-decr* (*j3* + 1)
definition *tmL* ≡ *WHILE* [] ; λ*rs. rs* ! (*j3* + 1) ≠ □ *DO tmB3 DONE*
definition *tm3* ≡ *tm2* ;; *tmL*
definition *tm4* ≡ *tm3* ;; *tm-setn* (*j3* + 2) *0*
definition *tm5* ≡ *tm4* ;; *tm-cr j3*

lemma *tm5-eq-tm-range*: *tm5* = *tm-range j1 j2 j3*

unfolding *tm-range-def tm5-def tm4-def tm3-def tmL-def tmB3-def tmB2-def tmB1-def tm2-def tm1-def*
by *simp*

context
fixes *tps0* :: *tape list* **and** *k start delta* :: *nat*
assumes *jk*: $k \geq j3 + 3$ $j1 \neq j2$ $0 < j1$ $0 < j2$ $0 < j3$ $j1 < j3$ $j2 < j3$ *length tps0* = *k*
and *tps0*:
tps0 ! *j1* = ($\lfloor start \rfloor_N$, *1*)
tps0 ! *j2* = ($\lfloor delta \rfloor_N$, *1*)
tps0 ! *j3* = ($\lfloor [] \rfloor_{NL}$, *1*)
tps0 ! (*j3* + *1*) = ($\lfloor 0 \rfloor_N$, *1*)
tps0 ! (*j3* + *2*) = ($\lfloor 0 \rfloor_N$, *1*)
begin

definition *tps1* ≡ *tps0*
[*j3* + *2* := ($\lfloor start \rfloor_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *14* + *3* ∗ *nlength start*
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
by (*tform tps*: *nlength-0 assms tps0 tps1-def jk*)

definition *tps2* ≡ *tps0*
[*j3* + *2* := ($\lfloor start \rfloor_N$, *1*),
j3 + *1* := ($\lfloor delta \rfloor_N$, *1*)]

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *28* + *3* ∗ *nlength start* + *3* ∗ *nlength delta*
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def*
proof (*tform tps*: *jk tps0 tps1-def tps2-def*)
show *ttt* = *14* + *3* ∗ *nlength start* + (*14* + *3* ∗ (*nlength delta* + *nlength 0*))
using *assms* **by** *simp*
qed

definition *tpsL t* ≡ *tps0*
[*j3* + *2* := ($\lfloor start + t \rfloor_N$, *1*),
j3 + *1* := ($\lfloor delta - t \rfloor_N$, *1*),
j3 := *nltape* [*start*..<*start* + *t*]]

lemma *tpsL-eq-tps2*: *tpsL 0* = *tps2*
using *tpsL-def tps2-def tps0 jk*
by (*metis* (*mono-tags*, *lifting*) *One-nat-def Suc-n-not-le-n add-cancel-left-right eq-imp-le list-update-id list-update-swap minus-nat.diff-0 nllength-Nil not-numeral-le-zero upt-conv-Nil*)

definition *tpsB1 t* ≡ *tps0*
[*j3* + *2* := ($\lfloor start + t \rfloor_N$, *1*),
j3 + *1* := ($\lfloor delta - t \rfloor_N$, *1*),
j3 := *nltape* ([*start*..<*start* + *t*] @ [*start* + *t*])]

lemma *tmB1* [*transforms-intros*]:
assumes *ttt* = *6* + *2* ∗ *nlength* (*start* + *t*)
shows *transforms tmB1* (*tpsL t*) *ttt* (*tpsB1 t*)
unfolding *tmB1-def*
proof (*tform tps*: *tpsL-def tpsB1-def jk*)
show *ttt* = *7* + *nllength* [*start*..<*start* + *t*] − *Suc* (*nllength* [*start*..<*start* + *t*]) + *2* ∗ *nlength* (*start* + *t*)
using *assms* **by** *simp*
qed

definition *tpsB2 t* ≡ *tps0*
[*j3* + *2* := ($\lfloor Suc\ (start + t) \rfloor_N$, *1*),
j3 + *1* := ($\lfloor delta - t \rfloor_N$, *1*),

j3 := *nltape* ([*start*..<*start* + *t*] @ [*start* + *t*])]

lemma *tmB2* [*transforms-intros*]:
assumes *ttt* = *11* + *4* ∗ *nlength* (*start* + *t*)
shows *transforms tmB2* (*tpsL t*) *ttt* (*tpsB2 t*)
unfolding *tmB2-def* **by** (*tform tps*: *tpsL-def tpsB1-def tpsB2-def jk time*: *assms*)

definition *tpsB3 t* ≡ *tps0*
[*j3* + *2* := (⌊*Suc* (*start* + *t*)⌋$_N$, *1*),
j3 + *1* := (⌊*delta* − *t* − *1*⌋$_N$, *1*),
j3 := *nltape* ([*start*..<*start* + *t*] @ [*start* + *t*])]

lemma *tmB3*:
assumes *ttt* = *19* + *4* ∗ *nlength* (*start* + *t*) + *2* ∗ *nlength* (*delta* − *t*)
shows *transforms tmB3* (*tpsL t*) *ttt* (*tpsB3 t*)
unfolding *tmB3-def* **by** (*tform tps*: *tpsL-def tpsB2-def tpsB3-def jk time*: *assms*)

lemma *tpsB3*: *tpsB3 t* ≡ *tpsL* (*Suc t*)
using *tpsB3-def tpsL-def* **by** *simp*

lemma *tmB3'* [*transforms-intros*]:
assumes *ttt* = *19* + *6* ∗ *nlength* (*start* + *delta*) **and** *t* < *delta*
shows *transforms tmB3* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
proof −
have *19* + *4* ∗ *nlength* (*start* + *t*) + *2* ∗ *nlength* (*delta* − *t*) ≤ *19* + *4* ∗ *nlength* (*start* + *t*) + *2* ∗ *nlength delta*
using *nlength-mono* **by** *simp*
also have ... ≤ *19* + *4* ∗ *nlength* (*start* + *delta*) + *2* ∗ *nlength delta*
using *assms*(*2*) *nlength-mono* **by** *simp*
also have ... ≤ *19* + *4* ∗ *nlength* (*start* + *delta*) + *2* ∗ *nlength* (*start* + *delta*)
using *nlength-mono* **by** *simp*
also have ... = *19* + *6* ∗ *nlength* (*start* + *delta*)
by *simp*
finally have *19* + *4* ∗ *nlength* (*start* + *t*) + *2* ∗ *nlength* (*delta* − *t*) ≤ *19* + *6* ∗ *nlength* (*start* + *delta*) **.**
then show *?thesis*
using *tpsB3 tmB3 transforms-monotone assms*(*1*) **by** *metis*
qed

lemma *tmL*:
assumes *ttt* = *delta* ∗ (*21* + *6* ∗ *nlength* (*start* + *delta*)) + *1*
shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsL delta*)
unfolding *tmL-def*
proof (*tform*)
have *read* (*tpsL t*) ! (*j3* + *1*) ≠ □ ⟷ *t* < *delta* **for** *t*
using *tpsL-def read-ncontents-eq-0 jk* **by** *auto*
then show ⋀*t*. *t* < *delta* ⟹ *read* (*tpsL t*) ! (*j3* + *1*) ≠ □ **and** ¬ *read* (*tpsL delta*) ! (*j3* + *1*) ≠ □
by *simp-all*
show *delta* ∗ (*19* + *6* ∗ *nlength* (*start* + *delta*) + *2*) + *1* ≤ *ttt*
using *assms*(*1*) **by** *simp*
qed

lemma *tmL'* [*transforms-intros*]:
assumes *ttt* = *delta* ∗ (*21* + *6* ∗ *nlength* (*start* + *delta*)) + *1*
shows *transforms tmL tps2 ttt* (*tpsL delta*)
using *tmL assms tpsL-eq-tps2* **by** *simp*

definition *tps3* ≡ *tps0*
[*j3* + *2* := (⌊*start* + *delta*⌋$_N$, *1*),
j3 := *nltape* [*start*..<*start* + *delta*]]

lemma *tpsL-eq-tps3*: *tpsL delta* = *tps3*
using *tps3-def tps0 jk tpsL-def*
by (*smt* (*verit*) *One-nat-def add-left-imp-eq cancel-comm-monoid-add-class.diff-cancel list-update-id*

list-update-swap n-not-Suc-n numeral-2-eq-2)

lemma *tm3*:
assumes *ttt* = *29* + *3* ∗ *nlength start* + *3* ∗ *nlength delta* + *delta* ∗ (*21* + *6* ∗ *nlength* (*start* + *delta*))
shows *transforms tm3 tps0 ttt* (*tpsL delta*)
unfolding *tm3-def* **by** (*tform time*: *assms*)

lemma *tm3′* [*transforms-intros*]:
assumes *ttt* = *29* + *3* ∗ *nlength start* + *3* ∗ *nlength delta* + *delta* ∗ (*21* + *6* ∗ *nlength* (*start* + *delta*))
shows *transforms tm3 tps0 ttt tps3*
using *assms tm3 tpsL-eq-tps3* **by** *simp*

definition *tps4* ≡ *tps0*
[*j3* := *nltape* [*start*..<*start* + *delta*]]

lemma *tm4*:
assumes *ttt* = *39* + *3* ∗ *nlength start* + *3* ∗ *nlength delta* + *delta* ∗ (*21* + *6* ∗ *nlength* (*start* + *delta*)) +
2 ∗ *nlength* (*start* + *delta*)
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform tps*: *tps3-def tps4-def jk time*: *assms*)
show *tps4* = *tps3*[*j3* + *2* := (⌊*0*⌋$_N$, *1*)]
using *tps4-def tps3-def tps0 jk*
by (*metis* (*mono-tags*, *lifting*) *Suc-neq-Zero add-cancel-right-right list-update-id list-update-overwrite*
list-update-swap numeral-2-eq-2)
qed

lemma *tm4′* [*transforms-intros*]:
assumes *ttt* = *Suc delta* ∗ (*39* + *8* ∗ *nlength* (*start* + *delta*))
shows *transforms tm4 tps0 ttt tps4*
proof −
let *?ttt* = *39* + *3* ∗ *nlength start* + *3* ∗ *nlength delta* + *delta* ∗ (*21* + *6* ∗ *nlength* (*start* + *delta*)) + *2* ∗ *nlength* (*start* + *delta*)
have *?ttt* ≤ *39* + *3* ∗ *nlength* (*start* + *delta*) + *3* ∗ *nlength* (*start* + *delta*) +
delta ∗ (*21* + *6* ∗ *nlength* (*start* + *delta*)) + *2* ∗ *nlength* (*start* + *delta*)
using *nlength-mono* **by** (*meson add-le-mono add-le-mono1 le-add2 nat-add-left-cancel-le nat-le-iff-add nat-mult-le-cancel-disj*)
also have ... = *39* + *8* ∗ *nlength* (*start* + *delta*) + *delta* ∗ (*21* + *6* ∗ *nlength* (*start* + *delta*))
by *simp*
also have ... ≤ *39* + *8* ∗ *nlength* (*start* + *delta*) + *delta* ∗ (*39* + *8* ∗ *nlength* (*start* + *delta*))
by *simp*
also have ... = *Suc delta* ∗ (*39* + *8* ∗ *nlength* (*start* + *delta*))
by *simp*
finally have *?ttt* ≤ *Suc delta* ∗ (*39* + *8* ∗ *nlength* (*start* + *delta*)) **.**
then show *?thesis*
using *assms tm4 transforms-monotone tps4-def* **by** *simp*
qed

definition *tps5* ≡ *tps0*
[*j3* := (⌊[*start*..<*start* + *delta*]⌋$_{NL}$, *1*)]

lemma *tm5*:
assumes *ttt* = *Suc delta* ∗ (*39* + *8* ∗ *nlength* (*start* + *delta*)) + *Suc* (*Suc* (*Suc* (*nllength* [*start*..<*start* + *delta*])))
shows *transforms tm5 tps0 ttt tps5*
unfolding *tm5-def*
proof (*tform tps*: *tps4-def tps5-def jk time*: *assms tps4-def jk*)
show *clean-tape* (*tps4* ! *j3*)
using *tps4-def jk clean-tape-nlcontents* **by** *simp*
qed

lemma *tm5′*:
assumes *ttt* = *Suc delta* ∗ (*43* + *9* ∗ *nlength* (*start* + *delta*))
shows *transforms tm5 tps0 ttt tps5*

proof −
  let *?ttt* = *Suc delta* ∗ (*39* + *8* ∗ *nlength* (*start* + *delta*)) + *Suc* (*Suc* (*Suc* (*nllength* [*start*..<*start* + *delta*])))
  have *nllength* [*start*..<*start* + *delta*] ≤ *Suc* (*nlength* (*start* + *delta*)) ∗ *delta*
    using *nllength-le-len-mult-max*[*of* [*start*..<*start* + *delta*] *start* + *delta*] **by** *simp*
  then have *?ttt* ≤ *Suc delta* ∗ (*39* + *8* ∗ *nlength* (*start* + *delta*)) + *3* + *Suc* (*nlength* (*start* + *delta*)) ∗ *delta*
    by *simp*
  also have ... ≤ *3* + *Suc delta* ∗ (*39* + *8* ∗ *nlength* (*start* + *delta*)) + *Suc delta* ∗ *Suc* (*nlength* (*start* + *delta*))
    by *simp*
  also have ... = *3* + *Suc delta* ∗ (*39* + *8* ∗ *nlength* (*start* + *delta*) + *Suc* (*nlength* (*start* + *delta*)))
    by *algebra*
  also have ... = *3* + *Suc delta* ∗ (*40* + *9* ∗ *nlength* (*start* + *delta*))
    by *simp*
  also have ... ≤ *Suc delta* ∗ (*43* + *9* ∗ *nlength* (*start* + *delta*))
    by *simp*
  finally have *?ttt* ≤ *Suc delta* ∗ (*43* + *9* ∗ *nlength* (*start* + *delta*)) **.**
  then show *?thesis*
    using *tm5 assms transforms-monotone* **by** *simp*
qed

end

end

lemma *transforms-tm-rangeI* [*transforms-intros*]:
  fixes *j1 j2 j3* :: *tapeidx*
  fixes *tps tps′* :: *tape list* **and** *k start delta* :: *nat*
  assumes *k* ≥ *j3* + *3 j1* ≠ *j2 0* < *j1 0* < *j2 j1* < *j3 j2* < *j3 length tps* = *k*
  assumes
    tps ! *j1* = ($\lfloor start \rfloor_N$, *1*)
    tps ! *j2* = ($\lfloor delta \rfloor_N$, *1*)
    tps ! *j3* = ($\lfloor [] \rfloor_{NL}$, *1*)
    tps ! (*j3* + *1*) = ($\lfloor 0 \rfloor_N$, *1*)
    tps ! (*j3* + *2*) = ($\lfloor 0 \rfloor_N$, *1*)
  assumes *ttt* = *Suc delta* ∗ (*43* + *9* ∗ *nlength* (*start* + *delta*))
  assumes *tps′* = *tps*
    [*j3* := ($\lfloor [start..<start + delta] \rfloor_{NL}$, *1*)]
  shows *transforms* (*tm-range j1 j2 j3*) *tps ttt tps′*
proof −
  interpret *loc*: *turing-machine-range j1 j2 j3* **.**
  show *?thesis*
    using *assms loc.tm5-eq-tm-range loc.tm5′ loc.tps5-def* **by** *simp*
qed

2.8.9 Creating singleton lists

The next Turing machine appends the symbol | to the symbols on tape j. Thus it turns a number into a singleton list containing this number.

definition *tm-to-list* :: *tapeidx* ⇒ *machine* **where**
  tm-to-list j ≡
    tm-right-until j {□} ;;
    tm-write j | ;;
    tm-cr j

lemma *tm-to-list-tm*:
  assumes *0* < *j* **and** *j* < *k* **and** *G* ≥ *5*
  shows *turing-machine k G* (*tm-to-list j*)
  unfolding *tm-to-list-def* **using** *tm-right-until-tm tm-write-tm tm-cr-tm assms* **by** *simp*

locale *turing-machine-to-list* =
  fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-right-until j* {□}

```
definition tm2 ≡ tm1 ;; tm-write j |
definition tm3 ≡ tm2 ;; tm-cr j

lemma tm3-eq-tm-to-list: tm3 = tm-to-list j
  using tm3-def tm2-def tm1-def tm-to-list-def by simp

context
  fixes tps0 :: tape list and k n :: nat
  assumes jk: 0 < j j < k length tps0 = k
    and tps0: tps0 ! j = (⌊n⌋N, 1)
begin

definition tps1 ≡ tps0[j := (⌊n⌋N, Suc (nlength n))]

lemma tm1 [transforms-intros]:
  assumes ttt = Suc (nlength n)
  shows transforms tm1 tps0 ttt tps1
  unfolding tm1-def
proof (tform tps: assms tps1-def tps0 jk)
  show rneigh (tps0 ! j) {0} (nlength n)
  proof (rule rneighI)
    show (tps0 ::: j) (tps0 :#: j + nlength n) ∈ {0}
      using tps0 jk by simp
    show ⋀n'. n' < nlength n ⟹ (tps0 ::: j) (tps0 :#: j + n') ∉ {0}
      using assms tps0 jk bit-symbols-canrepr contents-def by fastforce
  qed
qed

definition tps2 ≡ tps0[j := (⌊[n]⌋NL, Suc (nlength n))]

lemma tm2 [transforms-intros]:
  assumes ttt = Suc (Suc (nlength n))
  shows transforms tm2 tps0 ttt tps2
  unfolding tm2-def
proof (tform tps: assms tps1-def tps0 jk)
  have numlist [n] = canrepr n @ [|]
    using numlist-def by simp
  then show tps2 = tps1[j := tps1 ! j |:=| |]
    using assms tps1-def tps2-def tps0 jk numlist-def nlcontents-def contents-snoc
    by simp
qed

definition tps3 ≡ tps0[j := (⌊[n]⌋NL, 1)]

lemma tm3:
  assumes ttt = 5 + 2 * nlength n
  shows transforms tm3 tps0 ttt tps3
  unfolding tm3-def
proof (tform tps: tps2-def tps0 jk time: assms tps2-def jk)
  show clean-tape (tps2 ! j)
    using tps2-def jk clean-tape-nlcontents by simp
  show tps3 = tps2[j := tps2 ! j |#=| 1]
    using tps3-def tps2-def jk by simp
qed

end

end

lemma transforms-tm-to-listI [transforms-intros]:
  fixes j :: tapeidx
  fixes tps tps' :: tape list and ttt k n :: nat
  assumes 0 < j j < k length tps = k
```

assumes *tps* ! *j* = (⌊*n*⌋$_N$, *1*)
assumes *ttt* = *5* + *2* ∗ *nlength n*
assumes *tps′* = *tps*[*j* := (⌊[*n*]⌋$_{NL}$, *1*)]
shows *transforms* (*tm-to-list j*) *tps ttt tps′*
proof −
interpret *loc*: *turing-machine-to-list j* **.**
show *?thesis*
using *assms loc.tm3-eq-tm-to-list loc.tm3 loc.tps3-def* **by** *simp*
qed

2.8.10 Extending with a list

The next Turing machine extends the list on tape j_1 with the list on tape j_2. We assume that the tape head on j_1 is already at the end of the list.

definition *tm-extend* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-extend j1 j2 ≡ *tm-cp-until j2 j1* {□} ;; *tm-cr j2*

lemma *tm-extend-tm*:
assumes *0* < *j1* **and** *G* ≥ *4* **and** *j1* < *k* **and** *j2* < *k*
shows *turing-machine k G* (*tm-extend j1 j2*)
unfolding *tm-extend-def* **using** *assms tm-cp-until-tm tm-cr-tm* **by** *simp*

locale *turing-machine-extend* =
fixes *j1 j2* :: *tapeidx*
begin

definition *tm1* ≡ *tm-cp-until j2 j1* {□}
definition *tm2* ≡ *tm1* ;; *tm-cr j2*

lemma *tm2-eq-tm-extend*: *tm2* = *tm-extend j1 j2*
unfolding *tm2-def tm2-def tm1-def tm-extend-def* **by** *simp*

context
fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *ns1 ns2* :: *nat list*
assumes *jk*: *0* < *j1 j1* < *k j2* < *k j1* ≠ *j2 length tps0* = *k*
assumes *tps0*:
tps0 ! *j1* = *nltape ns1*
tps0 ! *j2* = (⌊*ns2*⌋$_{NL}$, *1*)
begin

definition *tps1* ≡ *tps0*
[*j1* := *nltape* (*ns1* @ *ns2*),
j2 := *nltape ns2*]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *Suc* (*nllength ns2*)
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
proof (*tform tps*: *tps1-def tps0 jk*)
let *?n* = *nllength ns2*
show *rneigh* (*tps0* ! *j2*) {*0*} *?n*
proof (*rule rneighI*)
show (*tps0* ::: *j2*) (*tps0* :#: *j2* + *nllength ns2*) ∈ {*0*}
using *tps0 nlcontents-def nllength-def jk* **by** *simp*
show ⋀*i*. *i* < *nllength ns2* ⟹ (*tps0* ::: *j2*) (*tps0* :#: *j2* + *i*) ∉ {*0*}
using *tps0 jk contents-def nlcontents-def nllength-def proper-symbols-numlist*
by *fastforce*
qed
show *ttt* = *Suc* (*nllength ns2*)
using *assms* **.**
show *tps1* = *tps0*
[*j2* := *tps0* ! *j2* |+| *nllength ns2*,
j1 := *implant* (*tps0* ! *j2*) (*tps0* ! *j1*) (*nllength ns2*)]

proof −
have *implant* (⌊*ns2*⌋$_{NL}$, *1*) (*nltape ns1*) (*nllength ns2*) = *nltape* (*ns1* @ *ns2*)
using *nlcontents-def nllength-def implant-contents* **by** (*simp add*: *numlist-append*)
moreover have *tps1* ! *j2* = *tps0* ! *j2* |+| *nllength ns2*
using *tps0 jk tps1-def* **by** *simp*
ultimately show *?thesis*
using *tps0 jk tps1-def* **by** (*metis fst-conv list-update-swap plus-1-eq-Suc snd-conv*)
qed
qed

definition *tps2* ≡ *tps0*[*j1* := *nltape* (*ns1* @ *ns2*)]

lemma *tm2*:
assumes *ttt* = *4* + *2* ∗ *nllength ns2*
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def*
proof (*tform tps*: *tps0 tps2-def tps1-def jk*)
show *clean-tape* (*tps1* ! *j2*)
using *tps1-def jk clean-tape-nlcontents* **by** *simp*
show *ttt* = *Suc* (*nllength ns2*) + (*tps1* :#: *j2* + *2*)
using *assms tps1-def jk* **by** *simp*
show *tps2* = *tps1*[*j2* := *tps1* ! *j2* |#=| *1*]
using *tps1-def jk tps2-def tps0*(*2*)
by (*metis fst-conv length-list-update list-update-id list-update-overwrite nth-list-update*)
qed

end

end

lemma *transforms-tm-extendI* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx*
fixes *tps tps′* :: *tape list* **and** *k* :: *nat* **and** *ns1 ns2* :: *nat list*
assumes *0* < *j1 j1* < *k j2* < *k j1* ≠ *j2 length tps* = *k*
assumes
tps ! *j1* = *nltape ns1*
tps ! *j2* = (⌊*ns2*⌋$_{NL}$, *1*)
assumes *ttt* = *4* + *2* ∗ *nllength ns2*
assumes *tps′* = *tps*[*j1* := *nltape* (*ns1* @ *ns2*)]
shows *transforms* (*tm-extend j1 j2*) *tps ttt tps′*
proof −
interpret *loc*: *turing-machine-extend j1 j2* **.**
show *?thesis*
using *loc.tm2-eq-tm-extend loc.tm2 loc.tps2-def assms* **by** *simp*
qed

An enhanced version of the previous Turing machine, the next one erases the list on tape j_2 after appending it to tape j_1.

definition *tm-extend-erase* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-extend-erase j1 j2 ≡ *tm-extend j1 j2* ;; *tm-erase-cr j2*

lemma *tm-extend-erase-tm*:
assumes *0* < *j1* **and** *0* < *j2* **and** *G* ≥ *4* **and** *j1* < *k* **and** *j2* < *k*
shows *turing-machine k G* (*tm-extend-erase j1 j2*)
unfolding *tm-extend-erase-def* **using** *assms tm-extend-tm tm-erase-cr-tm* **by** *simp*

lemma *transforms-tm-extend-eraseI* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx*
fixes *tps tps′* :: *tape list* **and** *k* :: *nat* **and** *ns1 ns2* :: *nat list*
assumes *0* < *j1 j1* < *k j2* < *k j1* ≠ *j2 0* < *j2 length tps* = *k*
assumes
tps ! *j1* = *nltape ns1*
tps ! *j2* = (⌊*ns2*⌋$_{NL}$, *1*)

```
  assumes ttt = 11 + 4 * nllength ns2
  assumes tps' = tps
    [j1 := nltape (ns1 @ ns2),
     j2 := (⌊[]⌋, 1)]
  shows transforms (tm-extend-erase j1 j2) tps ttt tps'
  unfolding tm-extend-erase-def
proof (tform tps: assms)
  let ?zs = numlist ns2
  show tps[j1 := nltape (ns1 @ ns2)] ::: j2 = ⌊?zs⌋
    using assms by (simp add: nlcontents-def)
  show proper-symbols ?zs
    using proper-symbols-numlist by simp
  show ttt = 4 + 2 * nllength ns2 +
      (tps[j1 := nltape (ns1 @ ns2)] :#: j2 + 2 * length (numlist ns2) + 6)
    using assms nllength-def by simp
qed
```

2.9 Lists of lists of numbers

In this section we introduce a representation for lists of lists of numbers as symbol sequences over the quaternary alphabet **01**|♯ and devise Turing machines for the few operations on such lists that we need. A tape containing such representations corresponds to a variable of type *nat list list*. A tape in the start configuration corresponds to the empty list of lists of numbers.

Many proofs in this section are copied from the previous section with minor modifications, mostly replacing the symbol | with ♯.

2.9.1 Representation as symbol sequence

We apply the same principle as for representing lists of numbers. To represent a list of lists of numbers, every element, which is now a list of numbers, is terminated by the symbol ♯. In this way the empty symbol sequence represents the empty list of lists of numbers. The list [[]] containing only an empty list is represented by ♯ and the list [[0]] containing only a list containing only a 0 is represented by |♯. As another example, the list $[[14, 0, 0, 7], [], [0], [25]]$ is represented by **0111**|||**111**|♯♯|♯**10011**|♯. The number of ♯ symbols equals the number of elements in the list.

```
definition numlistlist :: nat list list ⇒ symbol list where
  numlistlist nss ≡ concat (map (λns. numlist ns @ [♯]) nss)

lemma numlistlist-Nil: numlistlist [] = []
  using numlistlist-def by simp

proposition numlistlist [[]] = [♯]
  using numlistlist-def by (simp add: numlist-Nil)

lemma proper-symbols-numlistlist: proper-symbols (numlistlist nss)
proof (induction nss)
  case Nil
  then show ?case
    using numlistlist-def by simp
next
  case (Cons ns nss)
  have numlistlist (ns # nss) = numlist ns @ [♯] @ concat (map (λns. numlist ns @ [♯]) nss)
    using numlistlist-def by simp
  then have numlistlist (ns # nss) = numlist ns @ [♯] @ numlistlist nss
    using numlistlist-def by simp
  moreover have proper-symbols (numlist ns)
    using proper-symbols-numlist by simp
  moreover have proper-symbols [♯]
    by simp
  ultimately show ?case
```

using *proper-symbols-append Cons* **by** *presburger*
qed

lemma *symbols-lt-append*:
fixes *xs ys* :: *symbol list* **and** *z* :: *symbol*
assumes *symbols-lt z xs* **and** *symbols-lt z ys*
shows *symbols-lt z* (*xs* @ *ys*)
using *assms prop-list-append* **by** (*simp add*: *nth-append*)

lemma *symbols-lt-numlistlist*:
assumes $H \geq 6$
shows *symbols-lt H* (*numlistlist nss*)
proof (*induction nss*)
case *Nil*
then show *?case*
using *numlistlist-def* **by** *simp*
next
case (*Cons ns nss*)
have *numlistlist* (*ns* # *nss*) = *numlist ns* @ [♯] @ *concat* (*map* (λ*ns. numlist ns* @ [♯]) *nss*)
using *numlistlist-def* **by** *simp*
then have *numlistlist* (*ns* # *nss*) = *numlist ns* @ [♯] @ *numlistlist nss*
using *numlistlist-def* **by** *simp*
moreover have *symbols-lt H* (*numlist ns*)
using *assms numlist-234 nth-mem* **by** *fastforce*
moreover have *symbols-lt H* [♯]
using *assms* **by** *simp*
ultimately show *?case*
using *symbols-lt-append*[*of - H*] *Cons* **by** *presburger*
qed

lemma *symbols-lt-prefix-eq*:
assumes (*x* @ [♯]) @ *xs* = (*y* @ [♯]) @ *ys* **and** *symbols-lt 5 x* **and** *symbols-lt 5 y*
shows *x* = *y*
proof −
have ∗: *x* @ [♯] @ *xs* = *y* @ [♯] @ *ys*
(**is** *?lhs* = *?rhs*)
using *assms*(*1*) **by** *simp*
show *x* = *y*
proof (*cases length x* ≤ *length y*)
case *True*
then have *?lhs* ! *i* = *?rhs* ! *i* **if** *i* < *length x* **for** *i*
using *that* ∗ **by** *simp*
then have *eq*: *x* ! *i* = *y* ! *i* **if** *i* < *length x* **for** *i*
using *that True* **by** (*metis Suc-le-eq le-trans nth-append*)
have *?lhs* ! (*length x*) = ♯
by (*metis Cons-eq-appendI nth-append-length*)
then have *?rhs* ! (*length x*) = ♯
using ∗ **by** *metis*
moreover have *y* ! *i* ≠ ♯ **if** *i* < *length y* **for** *i*
using *that assms*(*3*) **by** *auto*
ultimately have *length y* ≤ *length x*
by (*metis linorder-le-less-linear nth-append*)
then have *length y* = *length x*
using *True* **by** *simp*
then show *?thesis*
using *eq* **by** (*simp add*: *list-eq-iff-nth-eq*)
next
case *False*
then have *?lhs* ! *i* = *?rhs* ! *i* **if** *i* < *length y* **for** *i*
using *that* ∗ **by** *simp*
have *?rhs* ! (*length y*) = ♯
by (*metis Cons-eq-appendI nth-append-length*)
then have *?lhs* ! (*length y*) = ♯

using $*$ **by** *metis*
then have x ! (*length* y) = ♯
using *False* **by** (*simp add*: *nth-append*)
then have *False*
using *assms*(*2*) *False*
by (*simp add*: *order-less-le*)
then show *?thesis*
by *simp*
qed
qed

lemma *numlistlist-inj*: *numlistlist ns1* = *numlistlist ns2* ⟹ *ns1* = *ns2*
proof (*induction ns1 arbitrary*: *ns2*)
case *Nil*
then show *?case*
using *numlistlist-def* **by** *simp*
next
case (*Cons n ns1*)
have *1*: *numlistlist* (*n* # *ns1*) = (*numlist n* @ [♯]) @ *numlistlist ns1*
using *numlistlist-def* **by** *simp*
then have *numlistlist ns2* = (*numlist n* @ [♯]) @ *numlistlist ns1*
using *Cons* **by** *simp*
then have *ns2* ≠ []
using *numlistlist-Nil* **by** *auto*
then have *2*: *ns2* = *hd ns2* # *tl ns2*
using ‹*ns2* ≠ []› **by** *simp*
then have *3*: *numlistlist ns2* = (*numlist* (*hd ns2*) @ [♯]) @ *numlistlist* (*tl ns2*)
using *numlistlist-def* **by** (*metis concat.simps*(*2*) *list.simps*(*9*))

have *4*: *hd ns2* = *n*
using *symbols-lt-prefix-eq 1 3 symbols-lt-numlist numlist-inj Cons* **by** *metis*
then have *numlistlist ns2* = (*numlist n* @ [♯]) @ *numlistlist* (*tl ns2*)
using *3* **by** *simp*
then have *numlistlist ns1* = *numlistlist* (*tl ns2*)
using *1* **by** (*simp add*: *Cons.prems*)
then have *ns1* = *tl ns2*
using *Cons* **by** *simp*
then show *?case*
using *2 4* **by** *simp*
qed

lemma *numlistlist-append*: *numlistlist* (*xs* @ *ys*) = *numlistlist xs* @ *numlistlist ys*
using *numlistlist-def* **by** *simp*

Similar to $\lfloor\text{-}\rfloor_N$ and $\lfloor\text{-}\rfloor_{NL}$, the tape contents for a list of list of numbers:

definition *nllcontents* :: *nat list list* ⇒ (*nat* ⇒ *symbol*) (‹$\lfloor\text{-}\rfloor_{NLL}$›) **where**
$\lfloor nss\rfloor_{NLL}$ ≡ ⌊*numlistlist nss*⌋

lemma *clean-tape-nllcontents*: *clean-tape* ($\lfloor ns\rfloor_{NLL}$, *i*)
by (*simp add*: *nllcontents-def proper-symbols-numlistlist*)

lemma *nllcontents-Nil*: $\lfloor[]\rfloor_{NLL}$ = ⌊[]⌋
using *nllcontents-def* **by** (*simp add*: *numlistlist-Nil*)

Similar to *nlength* and *nllength*, the length of the representation of a list of lists of numbers:

definition *nlllength* :: *nat list list* ⇒ *nat* **where**
nlllength nss ≡ *length* (*numlistlist nss*)

lemma *nlllength*: *nlllength nss* = (∑*ns*←*nss*. *Suc* (*nllength ns*))
using *nlllength-def numlistlist-def nllength-def* **by** (*induction nss*) *simp-all*

lemma *nlllength-Nil* [*simp*]: *nlllength* [] = *0*
using *nlllength-def numlistlist-def* **by** *simp*

```
lemma nlllength-Cons: nlllength (ns # nss) = Suc (nllength ns) + nlllength nss
  by (simp add: nlllength)

lemma length-le-nlllength: length nss ≤ nlllength nss
  using nlllength sum-list-mono[of nss λ-. 1 λns. Suc (nllength ns)] sum-list-const[of 1 nss]
  by simp

lemma member-le-nlllength-1: ns ∈ set nss ⟹ nllength ns ≤ nlllength nss − 1
  using nlllength by (induction nss) auto

lemma nlllength-take:
  assumes i < length nss
  shows nlllength (take i nss) + nllength (nss ! i) < nlllength nss
proof −
  have nss = take i nss @ [nss ! i] @ drop (Suc i) nss
    using assms by (metis Cons-eq-appendI append-self-conv2 id-take-nth-drop)
  then have numlistlist nss = numlistlist (take i nss) @ numlistlist [nss ! i] @ numlistlist (drop (Suc i) nss)
    using numlistlist-append by metis
  then have nlllength nss = nlllength (take i nss) + nlllength [nss ! i] + nlllength (drop (Suc i) nss)
    by (simp add: nlllength-def)
  then have nlllength nss ≥ nlllength (take i nss) + nlllength [nss ! i]
    by simp
  then have nlllength nss ≥ nlllength (take i nss) + Suc (nllength (nss ! i))
    using nlllength by simp
  then show ?thesis
    by simp
qed

lemma take-drop-numlistlist:
  assumes i < length ns
  shows take (Suc (nllength (ns ! i))) (drop (nlllength (take i ns)) (numlistlist ns)) = numlist (ns ! i) @ [♯]
proof −
  have numlistlist ns = numlistlist (take i ns) @ numlistlist (drop i ns)
    using numlistlist-append by (metis append-take-drop-id)
  moreover have numlistlist (drop i ns) = numlistlist [ns ! i] @ numlistlist (drop (Suc i) ns)
    using assms numlistlist-append by (metis Cons-nth-drop-Suc append-Cons self-append-conv2)
  ultimately have numlistlist ns = numlistlist (take i ns) @ numlistlist [ns ! i] @ numlistlist (drop (Suc i) ns)
    by simp
  then have drop (nlllength (take i ns)) (numlistlist ns) = numlistlist [ns ! i] @ numlistlist (drop (Suc i) ns)
    by (simp add: nlllength-def)
  then have drop (nlllength (take i ns)) (numlistlist ns) = numlist (ns ! i) @ [♯] @ numlistlist (drop (Suc i) ns)
    using numlistlist-def by simp
  then show ?thesis
    by (simp add: nllength-def)
qed

corollary take-drop-numlistlist':
  assumes i < length ns
  shows take (nllength (ns ! i)) (drop (nlllength (take i ns)) (numlistlist ns)) = numlist (ns ! i)
 using take-drop-numlistlist[OF assms] nllength-def by (metis append-assoc append-eq-conv-conj append-take-drop-id)

corollary numlistlist-take-at-term:
  assumes i < length ns
  shows numlistlist ns ! (nlllength (take i ns) + nllength (ns ! i)) = ♯
  using assms take-drop-numlistlist nlllength-def numlistlist-append
 by (smt (verit) append-eq-conv-conj append-take-drop-id lessI nllength-def nth-append-length nth-append-length-plus
nth-take)

lemma nlllength-take-Suc:
  assumes i < length ns
  shows nlllength (take i ns) + Suc (nllength (ns ! i)) = nlllength (take (Suc i) ns)
proof −
```

have *ns* = *take i ns* @ [*ns* ! *i*] @ *drop* (*Suc i*) *ns*
using *assms* **by** (*metis Cons-eq-appendI append-self-conv2 id-take-nth-drop*)
then have *numlistlist ns* = *numlistlist* (*take i ns*) @ *numlistlist* [*ns* ! *i*] @ *numlistlist* (*drop* (*Suc i*) *ns*)
using *numlistlist-append* **by** *metis*
then have *nlllength ns* = *nlllength* (*take i ns*) + *nlllength* [*ns* ! *i*] + *nlllength* (*drop* (*Suc i*) *ns*)
by (*simp add*: *nlllength-def*)
moreover have *nlllength ns* = *nlllength* (*take* (*Suc i*) *ns*) + *nlllength* (*drop* (*Suc i*) *ns*)
using *numlistlist-append* **by** (*metis append-take-drop-id length-append nlllength-def*)
ultimately have *nlllength* (*take* (*Suc i*) *ns*) = *nlllength* (*take i ns*) + *nlllength* [*ns* ! *i*]
by *simp*
then show *?thesis*
using *nlllength* **by** *simp*
qed

lemma *numlistlist-take-at*:
assumes *i* < *length ns* **and** *j* < *nllength* (*ns* ! *i*)
shows *numlistlist ns* ! (*nlllength* (*take i ns*) + *j*) = *numlist* (*ns* ! *i*) ! *j*
proof −
have *ns* = *take i ns* @ [*ns* ! *i*] @ *drop* (*Suc i*) *ns*
using *assms* **by** (*metis Cons-eq-appendI append-self-conv2 id-take-nth-drop*)
then have *numlistlist ns* = (*numlistlist* (*take i ns*) @ *numlistlist* [*ns* ! *i*]) @ *numlistlist* (*drop* (*Suc i*) *ns*)
using *numlistlist-append* **by** (*metis append-assoc*)
moreover have *nlllength* (*take i ns*) + *j* < *nlllength* (*take i ns*) + *nlllength* [*ns* ! *i*]
using *assms*(*2*) *nlllength* **by** *simp*
ultimately have *numlistlist ns* ! (*nlllength* (*take i ns*) + *j*) =
(*numlistlist* (*take i ns*) @ *numlistlist* [*ns* ! *i*]) ! (*nlllength* (*take i ns*) + *j*)
by (*metis length-append nlllength-def nth-append*)
also have ... = *numlistlist* [*ns* ! *i*] ! *j*
by (*simp add*: *nlllength-def*)
also have ... = (*numlist* (*ns* ! *i*) @ [♯]) ! *j*
using *numlistlist-def* **by** *simp*
also have ... = *numlist* (*ns* ! *i*) ! *j*
using *assms*(*2*) **by** (*metis nllength-def nth-append*)
finally show *?thesis* **.**
qed

lemma *nllcontents-rneigh-5*:
assumes *i* < *length ns*
shows *rneigh* ($\lfloor ns \rfloor_{NLL}$, *Suc* (*nlllength* (*take i ns*))) {♯} (*nllength* (*ns* ! *i*))
proof (*rule rneighI*)
let *?tp* = ($\lfloor ns \rfloor_{NLL}$, *Suc* (*nlllength* (*take i ns*)))
show *fst ?tp* (*snd ?tp* + *nllength* (*ns* ! *i*)) ∈ {♯}
proof −
have *snd ?tp* + *nllength* (*ns* ! *i*) ≤ *nlllength ns*
using *nlllength-take assms* **by** (*simp add*: *Suc-leI*)
then have *fst ?tp* (*snd ?tp* + *nllength* (*ns* ! *i*)) =
numlistlist ns ! (*nlllength* (*take i ns*) + *nllength* (*ns* ! *i*))
using *nllcontents-def contents-inbounds nlllength-def* **by** *simp*
then show *?thesis*
using *numlistlist-take-at-term assms* **by** *simp*
qed
show *fst ?tp* (*snd ?tp* + *j*) ∉ {♯} **if** *j* < *nllength* (*ns* ! *i*) **for** *j*
proof −
have *snd ?tp* + *nllength* (*ns* ! *i*) ≤ *nlllength ns*
using *nlllength-take assms* **by** (*simp add*: *Suc-leI*)
then have *snd ?tp* + *j* ≤ *nlllength ns*
using *that* **by** *simp*
then have *fst ?tp* (*snd ?tp* + *j*) = *numlistlist ns* ! (*nlllength* (*take i ns*) + *j*)
using *nllcontents-def contents-inbounds nlllength-def* **by** *simp*
then have *fst ?tp* (*snd ?tp* + *j*) = *numlist* (*ns* ! *i*) ! *j*
using *assms that numlistlist-take-at* **by** *simp*
moreover have *numlist* (*ns* ! *i*) ! *j* ≠ ♯
using *numlist-234 that nllength-def nth-mem* **by** *fastforce*

ultimately show *?thesis*
by *simp*
qed
qed

A tape containing a list of lists of numbers, with the tape head after all the symbols:

abbreviation *nlltape* :: *nat list list* ⇒ *tape* **where**
nlltape ns ≡ ($\lfloor ns \rfloor_{NLL}$, *Suc* (*nlllength ns*))

Like before but with the tape head or at the beginning of the i-th list element:

abbreviation *nlltape′* :: *nat list list* ⇒ *nat* ⇒ *tape* **where**
nlltape′ ns i ≡ ($\lfloor ns \rfloor_{NLL}$, *Suc* (*nlllength* (*take i ns*)))

lemma *nlltape′-0*: *nlltape′ ns 0* = ($\lfloor ns \rfloor_{NLL}$, *1*)
using *nlllength* **by** *simp*

lemma *nlltape′-tape-read*: |.| (*nlltape′ nss i*) = *0* ⟷ *i* ≥ *length nss*
proof −
have |.| (*nlltape′ nss i*) = *0* **if** *i* ≥ *length nss* **for** *i*
proof −
have *nlltape′ nss i* ≡ ($\lfloor nss \rfloor_{NLL}$, *Suc* (*nlllength nss*))
using *that* **by** *simp*
then show *?thesis*
using *nllcontents-def contents-outofbounds nlllength-def*
by (*metis Suc-eq-plus1 add.left-neutral fst-conv less-Suc0 less-add-eq-less snd-conv*)
qed
moreover have |.| (*nlltape′ nss i*) ≠ *0* **if** *i* < *length nss* **for** *i*
using *that nllcontents-def contents-inbounds nlllength-def nlllength-take proper-symbols-numlistlist*
by (*metis Suc-leI add-Suc-right diff-Suc-1 fst-conv less-add-same-cancel1 less-le-trans not-add-less2 plus-1-eq-Suc snd-conv zero-less-Suc*)
ultimately show *?thesis*
by (*meson le-less-linear*)
qed

2.9.2 Appending an element

The next Turing machine is very similar to *tm-append*, just with terminator symbol ♯ instead of |. It appends a list of numbers given on tape j_2 to the list of lists of numbers on tape j_1.

definition *tm-appendl* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-appendl j1 j2 ≡
tm-right-until j1 {□} ;;
tm-cp-until j2 j1 {□} ;;
tm-cr j2 ;;
tm-char j1 ♯

lemma *tm-appendl-tm*:
assumes *0* < *j1* **and** *G* ≥ *6* **and** *j1* < *k* **and** *j2* < *k*
shows *turing-machine k G* (*tm-appendl j1 j2*)
unfolding *tm-appendl-def* **using** *assms tm-right-until-tm tm-cp-until-tm tm-char-tm tm-cr-tm* **by** *simp*

locale *turing-machine-appendl* =
fixes *j1 j2* :: *tapeidx*
begin

definition *tm1* ≡ *tm-right-until j1* {□}
definition *tm2* ≡ *tm1* ;; *tm-cp-until j2 j1* {□}
definition *tm3* ≡ *tm2* ;; *tm-cr j2*
definition *tm4* ≡ *tm3* ;; *tm-char j1* ♯

lemma *tm4-eq-tm-append*: *tm4* = *tm-appendl j1 j2*
unfolding *tm4-def tm3-def tm2-def tm1-def tm-appendl-def* **by** *simp*

```
context
  fixes tps0 :: tape list and k i1 :: nat and ns :: nat list and nss :: nat list list
  assumes jk: length tps0 = k j1 < k j2 < k j1 ≠ j2 0 < j1
    and i1: i1 ≤ Suc (nlllength nss)
  assumes tps0:
      tps0 ! j1 = (⌊nss⌋NLL, i1)
      tps0 ! j2 = (⌊ns⌋NL, 1)
begin

definition tps1 ≡ tps0[j1 := nlltape nss]

lemma tm1 [transforms-intros]:
  assumes ttt = Suc (Suc (nlllength nss) − i1)
  shows transforms tm1 tps0 ttt tps1
  unfolding tm1-def
proof (tform tps: jk)
  let ?l = Suc (nlllength nss) − i1
  show rneigh (tps0 ! j1) {0} ?l
  proof (rule rneighI)
    show (tps0 ::: j1) (tps0 :#: j1 + ?l) ∈ {0}
      using tps0 jk nllcontents-def nlllength-def by simp
    show (tps0 ::: j1) (tps0 :#: j1 + i) ∉ {0} if i < Suc (nlllength nss) − i1 for i
    proof −
      have i1 + i < Suc (nlllength nss)
        using that i1 by simp
      then show ?thesis
        using proper-symbols-numlistlist nlllength-def tps0 nllcontents-def contents-def
        by (metis One-nat-def Suc-leI diff-Suc-1 fst-conv less-Suc-eq-0-disj less-nat-zero-code singletonD snd-conv)
    qed
  qed
  show ttt = Suc (Suc (nlllength nss) − i1)
    using assms(1) .
  show tps1 = tps0[j1 := tps0 ! j1 |+| Suc (nlllength nss) − i1]
    using tps1-def tps0 i1 by simp
qed

definition tps2 ≡ tps0
  [j1 := (⌊numlistlist nss @ numlist ns⌋, Suc (nlllength nss) + nllength ns),
   j2 := (⌊ns⌋NL, Suc (nllength ns))]

lemma tm2 [transforms-intros]:
  assumes ttt = 3 + nlllength nss − i1 + nllength ns
  shows transforms tm2 tps0 ttt tps2
  unfolding tm2-def
proof (tform tps: jk tps1-def)
  have j1: tps1 ! j1 = nlltape nss
    using tps1-def by (simp add: jk)
  have j2: tps1 ! j2 = (⌊ns⌋NL, 1)
    using tps1-def tps0 jk by simp
  show rneigh (tps1 ! j2) {0} (nllength ns)
  proof (rule rneighI)
    show (tps1 ::: j2) (tps1 :#: j2 + nllength ns) ∈ {0}
      using j2 by (simp add: nlcontents-def nllength-def)
    show ⋀i. i < nllength ns ⟹ (tps1 ::: j2) (tps1 :#: j2 + i) ∉ {0}
      using j2 tps0 contents-def nlcontents-def nllength-def proper-symbols-numlist by fastforce
  qed
  show tps2 = tps1
    [j2 := tps1 ! j2 |+| nllength ns,
     j1 := implant (tps1 ! j2) (tps1 ! j1) (nllength ns)]
    (is - = ?rhs)
  proof −
    have implant (tps1 ! j2) (tps1 ! j1) (nllength ns) = implant (⌊ns⌋NL, 1) (nlltape nss) (nllength ns)
      using j1 j2 by simp
```

also have ... =
($\lfloor$*numlistlist nss* @ (*take* (*nllength ns*) (*drop* (*1* − *1*) (*numlist ns*)))$\rfloor$,
Suc (*length* (*numlistlist nss*)) + *nllength ns*)
using *implant-contents nlcontents-def nllength-def nllcontents-def nlllength-def* **by** *simp*
also have ... = ($\lfloor$*numlistlist nss* @ *numlist ns*$\rfloor$, *Suc* (*length* (*numlistlist nss*)) + *nllength ns*)
by (*simp add*: *nllength-def*)
also have ... = ($\lfloor$*numlistlist nss* @ *numlist ns*$\rfloor$, *Suc* (*nlllength nss*) + *nllength ns*)
using *nlllength-def* **by** *simp*
also have ... = *tps2* ! *j1*
using *jk tps2-def* **by** (*metis nth-list-update-eq nth-list-update-neq*)
finally have *implant* (*tps1* ! *j2*) (*tps1* ! *j1*) (*nllength ns*) = *tps2* ! *j1* **.**
then have *tps2* ! *j1* = *implant* (*tps1* ! *j2*) (*tps1* ! *j1*) (*nllength ns*)
by *simp*
then have *tps2* ! *j1* = *?rhs* ! *j1*
using *tps1-def jk* **by** (*metis length-list-update nth-list-update-eq*)
moreover have *tps2* ! *j2* = *?rhs* ! *j2*
using *tps2-def tps1-def jk j2* **by** *simp*
moreover have *tps2* ! *j* = *?rhs* ! *j* **if** *j* < *length tps2 j* ≠ *j1 j* ≠ *j2* **for** *j*
using *that tps2-def tps1-def* **by** *simp*
moreover have *length tps2* = *length ?rhs*
using *tps1-def tps2-def* **by** *simp*
ultimately show *?thesis*
using *nth-equalityI* **by** *blast*
qed
show *ttt* = *Suc* (*Suc* (*nlllength nss*) − *i1*) + *Suc* (*nllength ns*)
using *assms*(*1*) *j1 tps0 i1* **by** *simp*
qed

definition *tps3* ≡ *tps0*
[*j1* := ($\lfloor$*numlistlist nss* @ *numlist ns*$\rfloor$, *Suc* (*nlllength nss*) + *nllength ns*)]

lemma *tm3* [*transforms-intros*]:
assumes *ttt* = *6* + *nlllength nss* − *i1* + *2* ∗ *nllength ns*
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def*
proof (*tform tps*: *jk i1 tps2-def*)
let *?tp1* = ($\lfloor$*numlistlist nss* @ *numlist ns*$\rfloor$, *Suc* (*nlllength nss* + *nllength ns*))
let *?tp2* = ($\lfloor$*ns*$\rfloor_{NL}$, *Suc* (*nllength ns*))
show *clean-tape* (*tps2* ! *j2*)
using *tps2-def jk* **by** (*simp add*: *clean-tape-nlcontents*)
show *tps3* = *tps2*[*j2* := *tps2* ! *j2* |#=| *1*]
using *tps3-def tps2-def jk tps0*(*2*)
by (*metis fst-eqD length-list-update list-update-overwrite list-update-same-conv nth-list-update*)
show *ttt* = *3* + *nlllength nss* − *i1* + *nllength ns* + (*tps2* :#: *j2* + *2*)
using *assms tps2-def jk tps0 i1* **by** *simp*
qed

definition *tps4* ≡ *tps0*
[*j1* := ($\lfloor$*numlistlist* (*nss* @ [*ns*])$\rfloor$, *Suc* (*nlllength* (*nss* @ [*ns*])))]

lemma *tm4*:
assumes *ttt* = *7* + *nlllength nss* − *i1* + *2* ∗ *nllength ns*
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform tps*: *jk tps3-def time*: *jk i1 assms*)
show *tps4* = *tps3*[*j1* := *tps3* ! *j1* |:=| ♯ |+| *1*]
(**is** *tps4* = *?tps*)
proof −
have *tps3* ! *j1* = ($\lfloor$*numlistlist nss* @ *numlist ns*$\rfloor$, *Suc* (*nlllength nss*) + *nllength ns*)
using *tps3-def jk* **by** *simp*
then have *?tps* = *tps0*[*j1* := ($\lfloor$*numlistlist nss* @ *numlist ns*$\rfloor$, *Suc* (*nlllength nss* + *nllength ns*)) |:=| ♯ |+| *1*]
using *tps3-def* **by** *simp*
moreover have ($\lfloor$*numlistlist nss* @ *numlist ns*$\rfloor$, *Suc* (*nlllength nss* + *nllength ns*)) |:=| ♯ |+| *1* =

```
      (⌊numlistlist (nss @ [ns])⌋, Suc (nlllength (nss @ [ns])))
    (is ?lhs = ?rhs)
  proof -
    have ?lhs =
      (⌊numlistlist nss @ numlist ns⌋(Suc (nlllength nss + nllength ns) := ♯),
       Suc (Suc (nlllength nss + nllength ns)))
      by simp
    also have ... =
      (⌊numlistlist nss @ numlist ns⌋(Suc (nlllength nss + nllength ns) := ♯),
       Suc (nlllength (nss @ [ns])))
      using nlllength-def numlistlist-def nllength-def numlist-def by simp
    also have ... = (⌊(numlistlist nss @ numlist ns) @ [♯]⌋, Suc (nlllength (nss @ [ns])))
      using contents-snoc nlllength-def nllength-def by (metis length-append)
    also have ... = (⌊numlistlist nss @ numlist ns @ [♯]⌋, Suc (nlllength (nss @ [ns])))
      by simp
    also have ... = (⌊numlistlist (nss @ [ns])⌋, Suc (nlllength (nss @ [ns])))
      using numlistlist-def by simp
    finally show ?lhs = ?rhs .
  qed
  ultimately show ?thesis
    using tps4-def by auto
 qed
qed

end

end

lemma transforms-tm-appendlI [transforms-intros]:
  fixes j1 j2 :: tapeidx
  fixes tps tps' :: tape list and ttt k i1 :: nat and ns :: nat list and nss :: nat list list
  assumes 0 < j1
  assumes length tps = k j1 < k j2 < k j1 ≠ j2
    and i1 ≤ Suc (nlllength nss)
  assumes
    tps ! j1 = (⌊nss⌋_NLL, i1)
    tps ! j2 = (⌊ns⌋_NL, 1)
  assumes ttt = 7 + nlllength nss - i1 + 2 * nllength ns
  assumes tps' = tps
    [j1 := nlltape (nss @ [ns])]
  shows transforms (tm-appendl j1 j2) tps ttt tps'
proof -
  interpret loc: turing-machine-appendl j1 j2 .
  show ?thesis
    using loc.tps4-def loc.tm4 loc.tm4-eq-tm-append assms nllcontents-def by simp
qed
```

2.9.3 Extending with a list

The next Turing machine extends a list of lists of numbers with another list of lists of numbers. It is in fact the same TM as for extending a list of numbers because in both cases extending means simply copying the contents from one tape to another. We introduce a new name for this TM and express the semantics in terms of lists of lists of numbers. The proof is almost the same except with *nllength* replaced by *nlllength* and so on.

```
definition tm-extendl :: tapeidx ⇒ tapeidx ⇒ machine where
  tm-extendl ≡ tm-extend

lemma tm-extendl-tm:
  assumes 0 < j1 and G ≥ 4 and j1 < k and j2 < k
  shows turing-machine k G (tm-extendl j1 j2)
  unfolding tm-extendl-def using assms tm-extend-tm by simp

locale turing-machine-extendl =
```

fixes *j1 j2* :: *tapeidx*
begin

definition *tm1* ≡ *tm-cp-until j2 j1* {□}
definition *tm2* ≡ *tm1* ;; *tm-cr j2*

lemma *tm2-eq-tm-extendl*: *tm2* = *tm-extendl j1 j2*
  unfolding *tm2-def tm2-def tm1-def tm-extendl-def tm-extend-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *nss1 nss2* :: *nat list list*
  assumes *jk*: *0* < *j1 j1* < *k j2* < *k j1* ≠ *j2 length tps0* = *k*
  assumes *tps0*:
    tps0 ! *j1* = *nlltape nss1*
    tps0 ! *j2* = ($\lfloor$*nss2*$\rfloor_{NLL}$, *1*)
begin

definition *tps1* ≡ *tps0*
  [*j1* := *nlltape* (*nss1* @ *nss2*),
   j2 := *nlltape nss2*]

lemma *tm1* [*transforms-intros*]:
  assumes *ttt* = *Suc* (*nlllength nss2*)
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def*
proof (*tform tps*: *tps1-def tps0 jk time*: *assms*)
  let *?n* = *nlllength nss2*
  show *rneigh* (*tps0* ! *j2*) {□} *?n*
  proof (*rule rneighI*)
    show (*tps0* ::: *j2*) (*tps0* :#: *j2* + *nlllength nss2*) ∈ {□}
      using *tps0 nllcontents-def nlllength-def jk* **by** *simp*
    show ⋀*i*. *i* < *nlllength nss2* ⟹ (*tps0* ::: *j2*) (*tps0* :#: *j2* + *i*) ∉ {□}
      using *tps0 jk contents-def nllcontents-def nlllength-def proper-symbols-numlistlist*
      by *fastforce*
  qed
  show *tps1* = *tps0*
    [*j2* := *tps0* ! *j2* |+| *nlllength nss2*,
     j1 := *implant* (*tps0* ! *j2*) (*tps0* ! *j1*) (*nlllength nss2*)]
  proof −
    have *implant* ($\lfloor$*nss2*$\rfloor_{NLL}$, *1*) (*nlltape nss1*) (*nlllength nss2*) = *nlltape* (*nss1* @ *nss2*)
      using *nllcontents-def nlllength-def implant-contents* **by** (*simp add*: *numlistlist-append*)
    moreover have *tps1* ! *j2* = *tps0* ! *j2* |+| *nlllength nss2*
      using *tps0 jk tps1-def* **by** *simp*
    ultimately show *?thesis*
      using *tps0 jk tps1-def* **by** (*metis fst-conv list-update-swap plus-1-eq-Suc snd-conv*)
  qed
qed

definition *tps2* ≡ *tps0*[*j1* := *nlltape* (*nss1* @ *nss2*)]

lemma *tm2*:
  assumes *ttt* = *4* + *2* ∗ *nlllength nss2*
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def*
proof (*tform tps*: *tps1-def tps2-def jk time*: *assms tps1-def jk*)
  show *clean-tape* (*tps1* ! *j2*)
    using *tps1-def jk clean-tape-nllcontents* **by** *simp*
  show *tps2* = *tps1*[*j2* := *tps1* ! *j2* |#=| *1*]
    using *tps1-def jk tps2-def tps0*(*2*)
    by (*metis fst-conv length-list-update list-update-id list-update-overwrite nth-list-update*)
qed

end

end

lemma *transforms-tm-extendlI* [*transforms-intros*]:
  fixes *j1 j2* :: *tapeidx*
  fixes *tps tps′* :: *tape list* **and** *k* :: *nat* **and** *nss1 nss2* :: *nat list list*
  assumes $0 < j1$ $j1 < k$ $j2 < k$ $j1 \neq j2$ *length tps* $= k$
  assumes
    tps ! *j1* = *nlltape nss1*
    tps ! *j2* = $(\lfloor nss2 \rfloor_{NLL},\ 1)$
  assumes *ttt* = $4 + 2 *$ *nlllength nss2*
  assumes *tps′* = *tps*[*j1* := *nlltape* (*nss1* @ *nss2*)]
  shows *transforms* (*tm-extendl j1 j2*) *tps ttt tps′*
proof −
  interpret *loc*: *turing-machine-extendl j1 j2* **.**
  show *?thesis*
    using *loc.tm2-eq-tm-extendl loc.tm2 loc.tps2-def assms* **by** *simp*
qed

The next Turing machine erases the appended list.

definition *tm-extendl-erase* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-extendl-erase j1 j2 ≡ *tm-extendl j1 j2* ;; *tm-erase-cr j2*

lemma *tm-extendl-erase-tm*:
  assumes $0 < j1$ **and** $0 < j2$ **and** $G \geq 4$ **and** $j1 < k$ **and** $j2 < k$
  shows *turing-machine k G* (*tm-extendl-erase j1 j2*)
  unfolding *tm-extendl-erase-def* **using** *assms tm-extendl-tm tm-erase-cr-tm* **by** *simp*

lemma *transforms-tm-extendl-eraseI* [*transforms-intros*]:
  fixes *tps tps′* :: *tape list* **and** *j1 j2* :: *tapeidx* **and** *ttt k* :: *nat* **and** *nss1 nss2* :: *nat list list*
  assumes $0 < j1$ $j1 < k$ $j2 < k$ $j1 \neq j2$ $0 < j2$ *length tps* $= k$
  assumes
    tps ! *j1* = *nlltape nss1*
    tps ! *j2* = $(\lfloor nss2 \rfloor_{NLL},\ 1)$
  assumes *ttt* = $11 + 4 *$ *nlllength nss2*
  assumes *tps′* = *tps*
    [*j1* := *nlltape* (*nss1* @ *nss2*),
     j2 := (⌊[]⌋, *1*)]
  shows *transforms* (*tm-extendl-erase j1 j2*) *tps ttt tps′*
  unfolding *tm-extendl-erase-def*
proof (*tform tps*: *assms*)
  let *?zs* = *numlistlist nss2*
  show *tps*[*j1* := *nlltape* (*nss1* @ *nss2*)] ::: *j2* = ⌊*?zs*⌋
    using *assms* **by** (*simp add*: *nllcontents-def*)
  show *proper-symbols ?zs*
    using *proper-symbols-numlistlist* **by** *simp*
  show *ttt* = $4 + 2 *$ *nlllength nss2* +
      (*tps*[*j1* := *nlltape* (*nss1* @ *nss2*)] :#: *j2* + $2 *$ *length* (*numlistlist nss2*) + *6*)
    using *assms nlllength-def* **by** *simp*
qed

2.9.4 Moving to the next element

Iterating over a list of lists of numbers works with the same Turing machine, *tm-nextract*, as for lists of numbers. We just have to set the parameter z to the terminator symbol ♯. For the proof we can also just copy from the previous section, replacing the symbol | by ♯ and *nllength* by *nlllength*, etc.

locale *turing-machine-nextract-5* =
  fixes *j1 j2* :: *tapeidx*
begin

definition *tm1* ≡ *tm-erase-cr j2*

definition *tm2* ≡ *tm1* ;; *tm-cp-until j1 j2* {♯}
definition *tm3* ≡ *tm2* ;; *tm-cr j2*
definition *tm4* ≡ *tm3* ;; *tm-right j1*

lemma *tm4-eq-tm-nextract*: *tm4* = *tm-nextract* ♯ *j1 j2*
  unfolding *tm1-def tm2-def tm3-def tm4-def tm-nextract-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *k idx* :: *nat* **and** *nss* :: *nat list list* **and** *dummy* :: *nat list*
  assumes *jk*: *j1* < *k j2* < *k 0* < *j1 0* < *j2 j1* ≠ *j2 length tps0* = *k*
    and *idx*: *idx* < *length nss*
    and *tps0*:
      tps0 ! *j1* = *nlltape′ nss idx*
      tps0 ! *j2* = (⌊*dummy*⌋$_{NL}$, *1*)
begin

definition *tps1* ≡ *tps0*[*j2* := (⌊[]⌋$_{NL}$, *1*)]

lemma *tm1* [*transforms-intros*]:
  assumes *ttt* = *7* + *2* ∗ *nllength dummy*
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def*
proof (*tform tps*: *jk idx tps0 tps1-def assms*)
  show *proper-symbols* (*numlist dummy*)
    using *proper-symbols-numlist* **by** *simp*
  show *tps1* = *tps0*[*j2* := (⌊[]⌋, *1*)]
    using *tps1-def* **by** (*simp add*: *nlcontents-def numlist-Nil*)
  show *tps0* ::: *j2* = ⌊*numlist dummy*⌋
    using *tps0* **by** (*simp add*: *nlcontents-def*)
  show *ttt* = *tps0* :#: *j2* + *2* ∗ *length* (*numlist dummy*) + *6*
    using *tps0 assms nllength-def* **by** *simp*
qed

definition *tps2* ≡ *tps0*
  [*j1* := (⌊*nss*⌋$_{NLL}$, *nlllength* (*take* (*Suc idx*) *nss*)),
   j2 := (⌊*nss* ! *idx*⌋$_{NL}$, *Suc* (*nllength* (*nss* ! *idx*)))]

lemma *tm2* [*transforms-intros*]:
  assumes *ttt* = *7* + *2* ∗ *nllength dummy* + *Suc* (*nllength* (*nss* ! *idx*))
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def*
proof (*tform tps*: *jk idx tps0 tps2-def tps1-def time*: *assms*)
  show *rneigh* (*tps1* ! *j1*) {♯} (*nllength* (*nss* ! *idx*))
    using *tps1-def tps0 jk* **by** (*simp add*: *idx nllcontents-rneigh-5*)
  show *tps2* = *tps1*
    [*j1* := *tps1* ! *j1* |+| *nllength* (*nss* ! *idx*),
     j2 := *implant* (*tps1* ! *j1*) (*tps1* ! *j2*) (*nllength* (*nss* ! *idx*))]
    (**is** *?l* = *?r*)
  proof (*rule nth-equalityI*)
    show *len*: *length ?l* = *length ?r*
      using *tps1-def tps2-def jk* **by** *simp*
    show *?l* ! *i* = *?r* ! *i* **if** *i* < *length ?l* **for** *i*
    proof −
      consider *i* = *j1* | *i* = *j2* | *i* ≠ *j1* ∧ *i* ≠ *j2*
        by *auto*
      then show *?thesis*
      proof (*cases*)
        case *1*
        then show *?thesis*
          using *tps0 that tps1-def tps2-def jk nlllength-take-Suc*[*OF idx*] *idx* **by** *simp*
      next
        case *2*
        then have *lhs*: *?l* ! *i* = (⌊*nss* ! *idx*⌋$_{NL}$, *Suc* (*nllength* (*nss* ! *idx*)))

```
        using tps2-def jk by simp
      let ?i = Suc (nlllength (take idx nss))
      have i1: ?i > 0
        by simp
      have nllength (nss ! idx) + (?i − 1) ≤ nlllength nss
        using idx nlllength-take by (metis add.commute diff-Suc-1 less-or-eq-imp-le)
      then have i2: nllength (nss ! idx) + (?i − 1) ≤ length (numlistlist nss)
        using nlllength-def by simp
      have ?r ! i = implant (tps1 ! j1) (tps1 ! j2) (nllength (nss ! idx))
        using 2 tps1-def jk by simp
      also have ... = implant (⌊nss⌋NLL, ?i) (⌊[]⌋NL, 1) (nllength (nss ! idx))
        using tps1-def jk tps0 by simp
      also have ... =
        (⌊[] @ (take (nllength (nss ! idx)) (drop (?i − 1) (numlistlist nss)))⌋,
         Suc (length []) + nllength (nss ! idx))
          using implant-contents[OF i1 i2] nllcontents-def nlcontents-def numlist-Nil by (metis One-nat-def
list.size(3))
      finally have ?r ! i =
        (⌊[] @ (take (nllength (nss ! idx)) (drop (?i − 1) (numlistlist nss)))⌋,
         Suc (length []) + nllength (nss ! idx)) .
      then have ?r ! i =
        (⌊take (nllength (nss ! idx)) (drop (nlllength (take idx nss)) (numlistlist nss))⌋,
         Suc (nllength (nss ! idx)))
        by simp
      then have ?r ! i = (⌊numlist (nss ! idx)⌋, Suc (nllength (nss ! idx)))
        using take-drop-numlistlist'[OF idx] by simp
      then show ?thesis
        using lhs nlcontents-def by simp
    next
      case 3
      then show ?thesis
        using that tps1-def tps2-def jk by simp
    qed
  qed
 qed
qed

definition tps3 ≡ tps0
  [j1 := (⌊nss⌋NLL, nlllength (take (Suc idx) nss)),
   j2 := (⌊nss ! idx⌋NL, 1)]

lemma tm3 [transforms-intros]:
  assumes ttt = 11 + 2 ∗ nllength dummy + 2 ∗ (nllength (nss ! idx))
  shows transforms tm3 tps0 ttt tps3
  unfolding tm3-def
proof (tform tps: jk idx tps0 tps2-def tps3-def assms)
  show clean-tape (tps2 ! j2)
    using tps2-def jk clean-tape-nlcontents by simp
qed

definition tps4 ≡ tps0
  [j1 := nlltape' nss (Suc idx),
   j2 := (⌊nss ! idx⌋NL, 1)]

lemma tm4:
  assumes ttt = 12 + 2 ∗ nllength dummy + 2 ∗ (nllength (nss ! idx))
  shows transforms tm4 tps0 ttt tps4
  unfolding tm4-def by (tform tps: jk idx tps0 tps3-def tps4-def assms)

end

end
```

lemma *transforms-tm-nextract-5I* [*transforms-intros*]:
  fixes *j1 j2* :: *tapeidx*
  fixes *tps tps'* :: *tape list* **and** *k idx* :: *nat* **and** *nss* :: *nat list list* **and** *dummy* :: *nat list*
  assumes $j1 < k$ $j2 < k$ $0 < j1$ $0 < j2$ $j1 \neq j2$ *length tps* = *k*
    and *idx* < *length nss*
  assumes
    tps ! *j1* = *nlltape' nss idx*
    tps ! *j2* = ($\lfloor dummy \rfloor_{NL}$, *1*)
  assumes *ttt* = *12* + *2* ∗ *nllength dummy* + *2* ∗ (*nllength* (*nss* ! *idx*))
  assumes *tps'* = *tps*
    [*j1* := *nlltape' nss* (*Suc idx*),
     j2 := ($\lfloor nss ! idx \rfloor_{NL}$, *1*)]
  shows *transforms* (*tm-nextract* ♯ *j1 j2*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-nextract-5 j1 j2* **.**
  show *?thesis*
    using *assms loc.tm4 loc.tps4-def loc.tm4-eq-tm-nextract* **by** *simp*
qed

end

2.10 Mapping between a binary and a quaternary alphabet

theory *Two-Four-Symbols*
  imports *Arithmetic*
begin

Functions are defined over bit strings. For Turing machines these bits are represented by the symbols **0** and **1**. Sometimes we want a TM to receive a pair of bit strings or output a list of numbers. Or we might want the TM to interpret the input as a list of lists of numbers. All these objects can naturally be represented over a four-symbol (quaternary) alphabet, as we have seen for pairs in Section 2.1.3 and for the lists in Sections 2.8 and 2.9.

To accommodate the aforementioned use cases, we define a straightforward mapping between the binary alphabet **01** and the quaternary alphabet **01**|♯ and devise Turing machines to encode and decode symbol sequences.

2.10.1 Encoding and decoding

The encoding maps:

0	↦	**00**
1	↦	**01**
|	↦	**10**
♯	↦	**11**

For example, the list $[6, 0, 1]$ is represented by the symbols **011**||**1**|, which is encoded as **00010110100110**. Pairing this sequence with the symbol sequence **0110** yields **00010110100110**♯**0110**, which is encoded as **0000000100010100010000010100110001010 0**.

definition *binencode* :: *symbol list* ⇒ *symbol list* **where**
  binencode ys ≡ *concat* (*map* (λ*y*. [*tosym* ((*y* − *2*) *div 2*), *tosym* ((*y* − *2*) *mod 2*)]) *ys*)

lemma *length-binencode* [*simp*]: *length* (*binencode ys*) = *2* ∗ *length ys*
  using *binencode-def* **by** (*induction ys*) *simp-all*

lemma *binencode-snoc*:
  binencode (*zs* @ [**0**]) = *binencode zs* @ [**0**, **0**]
  binencode (*zs* @ [**1**]) = *binencode zs* @ [**0**, **1**]
  binencode (*zs* @ [|]) = *binencode zs* @ [**1**, **0**]
  binencode (*zs* @ [♯]) = *binencode zs* @ [**1**, **1**]
  using *binencode-def* **by** *simp-all*

lemma *binencode-at-even*:
  assumes $i < length\ ys$
  shows $binencode\ ys\ !\ (2 * i) = 2 + (ys\ !\ i - 2)\ div\ 2$
  using *assms*
proof (*induction ys arbitrary*: *i*)
  case *Nil*
  then show *?case*
    by *simp*
next
  case (*Cons y ys*)
  have ∗: $binencode\ (y\ \#\ ys) = [2 + (y - 2)\ div\ 2,\ 2 + (y - 2)\ mod\ 2]\ @\ binencode\ ys$
    using *binencode-def* **by** *simp*
  show *?case*
  proof ($cases\ i = 0$)
    case *True*
    then show *?thesis*
      using ∗ **by** *simp*
  next
    case *False*
    then have $binencode\ (y\ \#\ ys)\ !\ (2 * i) = binencode\ ys\ !\ (2 * i - 2)$
      using ∗
      by (*metis One-nat-def length-Cons less-one list.size*(*3*) *mult-2 nat-mult-less-cancel-disj*
        nth-append numerals(*2*) *plus-1-eq-Suc*)
    also have $... = binencode\ ys\ !\ (2 * (i - 1))$
      using *False* **by** (*simp add*: *right-diff-distrib′*)
    also have $... = 2 + (ys\ !\ (i - 1) - 2)\ div\ 2$
      using *False Cons* **by** *simp*
    also have $... = 2 + ((y\ \#\ ys)\ !\ i - 2)\ div\ 2$
      using *False* **by** *simp*
    finally show *?thesis* **.**
  qed
qed

lemma *binencode-at-odd*:
  assumes $i < length\ ys$
  shows $binencode\ ys\ !\ (2 * i + 1) = 2 + (ys\ !\ i - 2)\ mod\ 2$
  using *assms*
proof (*induction ys arbitrary*: *i*)
  case *Nil*
  then show *?case*
    by *simp*
next
  case (*Cons y ys*)
  have ∗: $binencode\ (y\ \#\ ys) = [2 + (y - 2)\ div\ 2,\ 2 + (y - 2)\ mod\ 2]\ @\ binencode\ ys$
    using *binencode-def* **by** *simp*
  show *?case*
  proof ($cases\ i = 0$)
    case *True*
    then show *?thesis*
      using ∗ **by** *simp*
  next
    case *False*
    then have $binencode\ (y\ \#\ ys)\ !\ (2 * i + 1) = binencode\ ys\ !\ (2 * i + 1 - 2)$
      using ∗ **by** *simp*
    also have $... = binencode\ ys\ !\ (2 * (i - 1) + 1)$
      using *False*
        by (*metis Nat.add-diff-assoc2 One-nat-def Suc-leI diff-mult-distrib2 mult-2 mult-le-mono2 nat-1-add-1*
neq0-conv)
    also have $... = 2 + (ys\ !\ (i - 1) - 2)\ mod\ 2$
      using *False Cons* **by** *simp*
    also have $... = 2 + ((y\ \#\ ys)\ !\ i - 2)\ mod\ 2$
      using *False* **by** *simp*
    finally show *?thesis* **.**

  qed
qed

While *binencode* is defined for arbitrary symbol sequences, we only consider sequences over **01**|♯ to be binencodable.

abbreviation *binencodable* :: *symbol list* ⇒ *bool* **where**
  binencodable ys ≡ ∀ *i*<*length ys*. *2* ≤ *ys* ! *i* ∧ *ys* ! *i* < *6*

lemma *binencodable-append*:
  assumes *binencodable xs* **and** *binencodable ys*
  shows *binencodable* (*xs* @ *ys*)
  using *assms prop-list-append* **by** (*simp add*: *nth-append*)

lemma *bit-symbols-binencode*:
  assumes *binencodable ys*
  shows *bit-symbols* (*binencode ys*)
proof −
  have *2* ≤ *binencode ys* ! *i* ∧ *binencode ys* ! *i* ≤ *3* **if** *i* < *length* (*binencode ys*) **for** *i*
  proof (*cases even i*)
    case *True*
    then have *len*: *i div 2* < *length ys*
      using *length-binencode that* **by** *simp*
    moreover have *2* ∗ (*i div 2*) = *i*
      using *True* **by** *simp*
    ultimately have *binencode ys* ! *i* = *2* + (*ys* ! (*i div 2*) − *2*) *div 2*
      using *binencode-at-even*[*of i div 2 ys*] **by** *simp*
    moreover have *2* ≤ *ys* ! (*i div 2*) ∧ *ys* ! (*i div 2*) < *6*
      using *len assms* **by** *simp*
    ultimately show *?thesis*
      by *auto*
  next
    case *False*
    then have *len*: *i div 2* < *length ys*
      using *length-binencode that* **by** *simp*
    moreover have *2* ∗ (*i div 2*) + *1* = *i*
      using *False* **by** *simp*
    ultimately have *binencode ys* ! *i* = *2* + (*ys* ! (*i div 2*) − *2*) *mod 2*
      using *binencode-at-odd*[*of i div 2 ys*] **by** *simp*
    moreover have *2* ≤ *ys* ! (*i div 2*) ∧ *ys* ! (*i div 2*) < *6*
      using *len assms* **by** *simp*
    ultimately show *?thesis*
      by *simp*
  qed
  then show *?thesis*
    by *fastforce*
qed

An encoded symbol sequence is of even length. When decoding a symbol sequence of odd length, we ignore the last symbol. For example, **011100** and **0111001** are both decoded to **1**♯**0**.

The bit symbol sequence [*a*, *b*] is decoded to this symbol:

abbreviation *decsym* :: *symbol* ⇒ *symbol* ⇒ *symbol* **where**
  decsym a b ≡ *tosym* (*2* ∗ *todigit a* + *todigit b*)

definition *bindecode* :: *symbol list* ⇒ *symbol list* **where**
  bindecode zs ≡ *map* (λ*i*. *decsym* (*zs* ! (*2* ∗ *i*)) (*zs* ! (*Suc* (*2* ∗ *i*)))) [*0*..<*length zs div 2*]

lemma *length-bindecode* [*simp*]: *length* (*bindecode zs*) = *length zs div 2*
  using *bindecode-def* **by** *simp*

lemma *bindecode-at*:
  assumes *i* < *length zs div 2*
  shows *bindecode zs* ! *i* = *decsym* (*zs* ! (*2* ∗ *i*)) (*zs* ! (*Suc* (*2* ∗ *i*)))
  using *assms bindecode-def* **by** *simp*

```
lemma proper-bindecode: proper-symbols (bindecode zs)
  using bindecode-at by simp

lemma bindecode2345:
  assumes bit-symbols zs
  shows ∀ i<length (bindecode zs). bindecode zs ! i ∈ {2..<6}
  using assms bindecode-at by simp

lemma bindecode-odd:
  assumes length zs = 2 ∗ n + 1
  shows bindecode zs = bindecode (take (2 ∗ n) zs)
proof −
  have 1: take (2 ∗ n) zs ! (2 ∗ i) = zs ! (2 ∗ i) if i < n for i
    using assms that by simp
  have 2: take (2 ∗ n) zs ! (Suc (2 ∗ i)) = zs ! (Suc (2 ∗ i)) if i < n for i
    using assms that by simp
  have bindecode (take (2 ∗ n) zs) =
      map (λi. decsym ((take (2 ∗ n) zs) ! (2 ∗ i)) ((take (2 ∗ n) zs) ! (Suc (2 ∗ i)))) [0..<length (take (2 ∗ n)
zs) div 2]
    using bindecode-def by simp
  also have ... = map (λi. decsym ((take (2 ∗ n) zs) ! (2 ∗ i)) ((take (2 ∗ n) zs) ! (Suc (2 ∗ i)))) [0..<n]
    using assms by (simp add: min-absorb2)
  also have ... = map (λi. decsym (zs ! (2 ∗ i)) (zs ! (Suc (2 ∗ i)))) [0..<n]
    by simp
  also have ... = map (λi. decsym (zs ! (2 ∗ i)) (zs ! (Suc (2 ∗ i)))) [0..<length zs div 2]
    using assms by simp
  also have ... = bindecode zs
    using bindecode-def by simp
  finally show ?thesis
    by simp
qed

lemma bindecode-append:
  assumes even (length ys) and even (length zs)
  shows bindecode (ys @ zs) = bindecode ys @ bindecode zs
    (is ?lhs = ?rhs)
proof (rule nth-equalityI)
  show 1: length ?lhs = length ?rhs
    using assms by simp
  show ?lhs ! i = ?rhs ! i if i < length ?lhs for i
  proof (cases i < length ys div 2)
    case True
    have ?lhs ! i = decsym ((ys @ zs) ! (2 ∗ i)) ((ys @ zs) ! (Suc (2 ∗ i)))
      using that bindecode-at length-bindecode by simp
    also have ... = decsym (ys ! (2 ∗ i)) ((ys @ zs) ! (Suc (2 ∗ i)))
      using True assms(1)
      by (metis Suc-1 dvd-mult-div-cancel nat-mult-less-cancel-disj nth-append zero-less-Suc)
    also have ... = decsym (ys ! (2 ∗ i)) (ys ! (Suc (2 ∗ i)))
      using True assms(1)
      by (metis Suc-1 Suc-lessI Suc-mult-less-cancel1 dvd-mult-div-cancel dvd-triv-left even-Suc nth-append)
    also have ... = bindecode ys ! i
      using True bindecode-at by simp
    also have ... = ?rhs ! i
      by (simp add: True nth-append)
    finally show ?thesis .
  next
    case False
    let ?l = length ys
    have l: length (bindecode ys) = ?l div 2
      by simp
    have ?lhs ! i = decsym ((ys @ zs) ! (2 ∗ i)) ((ys @ zs) ! (Suc (2 ∗ i)))
      using that bindecode-at length-bindecode by simp
```

```
    also have ... = decsym (zs ! (2 * i − ?l)) ((ys @ zs) ! (Suc (2 * i)))
      using False assms
      by (metis dvd-mult-div-cancel nat-mult-less-cancel-disj nth-append)
    also have ... = decsym (zs ! (2 * i − ?l)) (zs ! (Suc (2 * i) − ?l))
      using False assms
     by (metis (no-types, lifting) Suc-eq-plus1 add-lessD1 dvd-mult-div-cancel nat-mult-less-cancel-disj nth-append)
    also have ... = bindecode zs ! (i − ?l div 2)
      using False bindecode-at that assms 1
      by simp (metis Nat.add-diff-assoc add-mono-thms-linordered-semiring(1)
          dvd-mult-div-cancel le-less-linear mult-2 plus-1-eq-Suc right-diff-distrib')
    also have ... = (bindecode ys @ bindecode zs) ! i
      using l False by (simp add: nth-append)
    finally show ?thesis .
  qed
qed

lemma bindecode-take-snoc:
  assumes i < length zs div 2
  shows bindecode (take (2 * i) zs) @ [decsym (zs ! (2*i)) (zs ! (Suc (2*i)))] = bindecode (take (2 * Suc i) zs)
proof −
  let ?ys = take (2 * i) zs
  let ?zs = take 2 (drop (2 * i) zs)
  have ?zs ! 0 = zs ! (2 * i)
    using assms by simp
  moreover have ?zs ! 1 = zs ! (Suc (2 * i))
    using assms by simp
  moreover have length ?zs = 2
    using assms by simp
  ultimately have ?zs = [zs ! (2 * i), zs ! (Suc (2 * i))]
    by (metis (no-types, lifting) Cons-nth-drop-Suc One-nat-def Suc-1 diff-Suc-1 drop0
      length-0-conv length-drop lessI list.inject zero-less-Suc)
  moreover have take (2 * i + 2) zs = ?ys @ ?zs
    using assms take-add by blast
  ultimately have take (2 * Suc i) zs = ?ys @ [zs ! (2 * i), zs ! (Suc (2 * i))]
    by simp
  moreover have even (length ?ys)
  proof −
    have length ?ys = 2 * i
      using assms by simp
    then show ?thesis
      by simp
  qed
  ultimately have bindecode (take (2 * Suc i) zs) = bindecode ?ys @ bindecode [zs ! (2 * i), zs ! (Suc (2 * i))]
    using bindecode-append by simp
  then show ?thesis
    using bindecode-def by simp
qed

lemma bindecode-encode:
  assumes binencodable ys
  shows bindecode (binencode ys) = ys
proof (rule nth-equalityI)
  show 1: length (bindecode (binencode ys)) = length ys
    using binencode-def bindecode-def by fastforce
  show bindecode (binencode ys) ! i = ys ! i if i < length (bindecode (binencode ys)) for i
  proof −
    have len: i < length ys
      using that 1 by simp
    let ?zs = binencode ys
    have i < length ?zs div 2
      using len by simp
    then have bindecode ?zs ! i = decsym (?zs ! (2 * i)) (?zs ! (Suc (2 * i)))
      using bindecode-def by simp
```

```
    also have ... = decsym (2 + (ys ! i − 2) div 2) (2 + (ys ! i − 2) mod 2)
      using binencode-at-even[OF len] binencode-at-odd[OF len] by simp
    also have ... = ys ! i
    proof −
      have ys ! i = 2 ∨ ys ! i = 3 ∨ ys ! i = 4 ∨ ys ! i = 5
        using assms len
        by (smt (verit) Suc-1 add-Suc-shift add-cancel-right-left eval-nat-numeral(3)
          less-Suc-eq numeral-3-eq-3 numeral-Bit0 verit-comp-simplify1(3))
      then show ?thesis
        by auto
    qed
    finally show bindecode ?zs ! i = ys ! i .
  qed
qed

lemma binencode-decode:
  assumes bit-symbols zs and even (length zs)
  shows binencode (bindecode zs) = zs
proof (rule nth-equalityI)
  let ?ys = bindecode zs
  show 1: length (binencode ?ys) = length zs
    using binencode-def bindecode-def assms(2) by fastforce
  show binencode ?ys ! i = zs ! i if i < length (binencode ?ys) for i
  proof −
    have ilen: i < length zs
      using 1 that by simp
    show ?thesis
    proof (cases even i)
      case True
      let ?j = i div 2
      have jlen: ?j < length zs div 2
        using ilen by (simp add: assms(2) less-mult-imp-div-less)
      then have ysj: ?ys ! ?j = decsym (zs ! (2 ∗ ?j)) (zs ! (Suc (2 ∗ ?j)))
        using bindecode-def by simp
      have binencode ?ys ! i = binencode ?ys ! (2 ∗ ?j)
        by (simp add: True)
      also have ... = 2 + (?ys ! ?j − 2) div 2
        using binencode-at-even jlen by simp
      also have ... = zs ! (2 ∗ ?j)
        using ysj True assms(1) ilen by auto
      also have ... = zs ! i
        using True by simp
      finally show binencode ?ys ! i = zs ! i .
    next
      case False
      let ?j = i div 2
      have jlen: ?j < length zs div 2
        using ilen by (simp add: assms(2) less-mult-imp-div-less)
      then have ysj: ?ys ! ?j = decsym (zs ! (2 ∗ ?j)) (zs ! (Suc (2 ∗ ?j)))
        using bindecode-def by simp
      have binencode ?ys ! i = binencode ?ys ! (2 ∗ ?j + 1)
        by (simp add: False)
      also have ... = 2 + (?ys ! ?j − 2) mod 2
        using binencode-at-odd jlen by simp
      also have ... = zs ! (2 ∗ ?j + 1)
        using ysj False assms(1) ilen by auto
      also have ... = zs ! i
        using False by simp
      finally show binencode ?ys ! i = zs ! i .
    qed
  qed
qed
```

```
lemma binencode-inj:
  assumes binencodable xs and binencodable ys and binencode xs = binencode ys
  shows xs = ys
  using assms bindecode-encode by metis
```

2.10.2 Turing machines for encoding and decoding

The next Turing machine implements *binencode* for *binencodable* symbol sequences. It expects a symbol sequence *zs* on tape j_1 and writes *binencode zs* to tape j_2.

```
definition tm-binencode :: tapeidx ⇒ tapeidx ⇒ machine where
  tm-binencode j1 j2 ≡
    WHILE [] ; λrs. rs ! j1 ≠ □ DO
      IF λrs. rs ! j1 = 0 THEN
        tm-print j2 [0, 0]
      ELSE
        IF λrs. rs ! j1 = 1 THEN
          tm-print j2 [0, 1]
        ELSE
          IF λrs. rs ! j1 = | THEN
            tm-print j2 [1, 0]
          ELSE
            tm-print j2 [1, 1]
          ENDIF
        ENDIF
      ENDIF ;;
      tm-right j1
    DONE

lemma tm-binencode-tm:
  assumes k ≥ 2 and G ≥ 4 and j1 < k and j2 < k and 0 < j2
  shows turing-machine k G (tm-binencode j1 j2)
proof −
  have ∗: symbols-lt G [0, 0] symbols-lt G [0, 1] symbols-lt G [1, 0] symbols-lt G [1, 1]
    using assms(2) by (simp-all add: nth-Cons′)
  then have turing-machine k G (tm-print j2 [0, 0])
    using tm-print-tm[OF assms(5,4,2)] assms by blast
  moreover have turing-machine k G (tm-print j2 [0, 1])
    using ∗ tm-print-tm[OF assms(5,4,2)] assms by blast
  moreover have turing-machine k G (tm-print j2 [1, 0])
    using ∗ tm-print-tm[OF assms(5,4,2)] assms by blast
  moreover have turing-machine k G (tm-print j2 [1, 1])
    using ∗ tm-print-tm[OF assms(5,4,2)] assms by blast
  ultimately show ?thesis
    unfolding tm-binencode-def
    using turing-machine-loop-turing-machine Nil-tm turing-machine-branch-turing-machine tm-right-tm assms
    by simp
qed

locale turing-machine-binencode =
  fixes j1 j2 :: tapeidx
begin

definition tm1 ≡ IF λrs. rs ! j1 = | THEN tm-print j2 [1, 0] ELSE tm-print j2 [1, 1] ENDIF
definition tm2 ≡ IF λrs. rs ! j1 = 1 THEN tm-print j2 [0, 1] ELSE tm1 ENDIF
definition tm3 ≡ IF λrs. rs ! j1 = 0 THEN tm-print j2 [0, 0] ELSE tm2 ENDIF
definition tm4 ≡ tm3 ;; tm-right j1
definition tm5 ≡ WHILE [] ; λrs. rs ! j1 ≠ □ DO tm4 DONE

lemma tm5-eq-tm-binencode: tm5 = tm-binencode j1 j2
  using tm5-def tm4-def tm3-def tm2-def tm1-def tm-binencode-def by simp

context
  fixes k :: nat and tps0 :: tape list and zs :: symbol list
```

assumes *jk*: $k = length\ tps0\ j1 \neq j2\ 0 < j2\ j1 < k\ j2 < k$
assumes *zs*: *binencodable zs*
assumes *tps0*:
$tps0\ !\ j1 = (\lfloor zs \rfloor,\ 1)$
$tps0\ !\ j2 = (\lfloor [] \rfloor,\ 1)$
begin

definition *tpsL* :: $nat \Rightarrow tape\ list$ **where**
$tpsL\ t \equiv tps0$
$[j1 := (\lfloor zs \rfloor,\ Suc\ t),$
$j2 := (\lfloor binencode\ (take\ t\ zs) \rfloor,\ Suc\ (2 * t))]$

lemma *tpsL-0*: $tpsL\ 0 = tps0$
unfolding *tpsL-def* **using** *tps0 jk*
by (*metis One-nat-def length-0-conv length-binencode list-update-id mult-0-right take0*)

definition *tpsL1* :: $nat \Rightarrow tape\ list$ **where**
$tpsL1\ t \equiv tps0$
$[j1 := (\lfloor zs \rfloor,\ Suc\ t),$
$j2 := (\lfloor binencode\ (take\ (Suc\ t)\ zs) \rfloor,\ Suc\ (2 * t) + 2)]$

lemma *read-tpsL*:
assumes $t < length\ zs$
shows $read\ (tpsL\ t)\ !\ j1 = zs\ !\ t$
proof −
have $tpsL\ t\ !\ j1 = (\lfloor zs \rfloor,\ Suc\ t)$
using *tpsL-def jk* **by** *simp*
moreover have $j1 < length\ (tpsL\ t)$
using *tpsL-def jk* **by** *simp*
ultimately show $read\ (tpsL\ t)\ !\ j1 = zs\ !\ t$
using *assms tapes-at-read′*
by (*metis Suc-leI contents-inbounds diff-Suc-1 fst-conv snd-conv zero-less-Suc*)
qed

lemma *tm1* [*transforms-intros*]:
assumes $t < length\ zs$ **and** $zs\ !\ t = | \vee zs\ !\ t = \sharp$
shows *transforms tm1* $(tpsL\ t)\ 4\ (tpsL1\ t)$
unfolding *tm1-def*
proof (*tform tps*: *jk tpsL-def*)
have *1*: $tpsL\ t\ !\ j2 = (\lfloor binencode\ (take\ t\ zs) \rfloor,\ Suc\ (2 * t))$
using *tpsL-def jk* **by** *simp*
have *2*: $length\ (binencode\ (take\ t\ zs)) = 2 * t$
using *assms*(*1*) **by** *simp*
have $inscribe\ (tpsL\ t\ !\ j2)\ [\mathbf{1},\ \mathbf{0}] = (\lfloor binencode\ (take\ t\ zs)\ @\ [\mathbf{1},\ \mathbf{0}] \rfloor,\ Suc\ (2 * t) + 2)$
using *inscribe-contents 1 2*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-eq-plus1 add-Suc-shift list.size*(*3*) *list.size*(*4*))
moreover have $binencode\ (take\ t\ zs)\ @\ [\mathbf{1},\ \mathbf{0}] = binencode\ (take\ (Suc\ t)\ zs)$ **if** $zs\ !\ t = |$
using *that assms*(*1*) *binencode-snoc* **by** (*metis take-Suc-conv-app-nth*)
ultimately have *5*: $inscribe\ (tpsL\ t\ !\ j2)\ [\mathbf{1},\ \mathbf{0}] = (\lfloor binencode\ (take\ (Suc\ t)\ zs) \rfloor,\ Suc\ (2 * t) + 2)$
if $zs\ !\ t = |$
using *that* **by** *simp*
have $inscribe\ (tpsL\ t\ !\ j2)\ [\mathbf{1},\ \mathbf{1}] = (\lfloor binencode\ (take\ t\ zs)\ @\ [\mathbf{1},\ \mathbf{1}] \rfloor,\ Suc\ (2 * t) + 2)$
using *inscribe-contents 1 2*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-eq-plus1 add-Suc-shift list.size*(*3*) *list.size*(*4*))
moreover have $binencode\ (take\ t\ zs)\ @\ [\mathbf{1},\ \mathbf{1}] = binencode\ (take\ (Suc\ t)\ zs)$ **if** $zs\ !\ t = 5$
using *that assms*(*1*) *binencode-snoc* **by** (*metis take-Suc-conv-app-nth*)
ultimately have *6*: $inscribe\ (tpsL\ t\ !\ j2)\ [\mathbf{1},\ \mathbf{1}] = (\lfloor binencode\ (take\ (Suc\ t)\ zs) \rfloor,\ Suc\ (2 * t) + 2)$
if $zs\ !\ t = \sharp$
using *that* **by** *simp*
have *7*: $tpsL1\ t = (tpsL\ t)[j2 := (\lfloor binencode\ (take\ (Suc\ t)\ zs) \rfloor,\ Suc\ (2 * t) + 2)]$
unfolding *tpsL1-def tpsL-def* **by** (*simp only*: *list-update-overwrite*)
then have *8*: $tpsL1\ t = (tpsL\ t)[j2 := inscribe\ (tpsL\ t\ !\ j2)\ [\mathbf{1},\ \mathbf{0}]]$ **if** $zs\ !\ t = |$
using *that 5* **by** *simp*

have *9*: *tpsL1 t* = (*tpsL t*)[*j2* := *inscribe* (*tpsL t* ! *j2*) [**1**, **1**]] **if** *zs* ! *t* = ♯
    using *that 6 7* **by** *simp*
  show *read* (*tpsL t*) ! *j1* = | ⟹
      tpsL1 t = (*tpsL t*)[*j2* := *inscribe* (*tpsL t* ! *j2*) [**1**, **0**]]
    using *read-tpsL*[*OF assms*(*1*)] *8* **by** *simp*
  show *read* (*tpsL t*) ! *j1* ≠ | ⟹
      tpsL1 t = (*tpsL t*)[*j2* := *inscribe* (*tpsL t* ! *j2*) [**1**, **1**]]
    using *read-tpsL*[*OF assms*(*1*)] *9 assms*(*2*) **by** *simp*
qed

lemma *tm2* [*transforms-intros*]:
  assumes *t* < *length zs* **and** *zs* ! *t* = | ∨ *zs* ! *t* = ♯ ∨ *zs* ! *t* = **1**
  shows *transforms tm2* (*tpsL t*) *5* (*tpsL1 t*)
  unfolding *tm2-def*
proof (*tform tps*: *tpsL-def jk assms*(*1*))
  have *1*: *tpsL t* ! *j2* = (⌊*binencode* (*take t zs*)⌋, *Suc* (*2* ∗ *t*))
    using *tpsL-def jk* **by** *simp*
  have *2*: *length* (*binencode* (*take t zs*)) = *2* ∗ *t*
    using *assms*(*1*) **by** *simp*
  show *read* (*tpsL t*) ! *j1* ≠ **1** ⟹ *zs* ! *t* = | ∨ *zs* ! *t* = ♯
    using *read-tpsL*[*OF assms*(*1*)] *assms*(*2*) **by** *simp*
  show *tpsL1 t* = (*tpsL t*)[*j2* := *inscribe* (*tpsL t* ! *j2*) [**0**, **1**]] **if** *read* (*tpsL t*) ! *j1* = **1**
  proof −
    have ∗: *zs* ! *t* = **1**
      using *read-tpsL*[*OF assms*(*1*)] *that* **by** *simp*
    have *inscribe* (*tpsL t* ! *j2*) [**0**, **1**] = (⌊*binencode* (*take t zs*) @ [*2*, **1**]⌋, *Suc* (*2* ∗ *t*) + *2*)
      using *inscribe-contents 1 2*
      by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-eq-plus1 add-Suc-shift list.size*(*3*) *list.size*(*4*))
    moreover have *binencode* (*take t zs*) @ [**0**, **1**] = *binencode* (*take* (*Suc t*) *zs*)
      using ∗ *assms*(*1*) *binencode-snoc* **by** (*metis take-Suc-conv-app-nth*)
    ultimately have *inscribe* (*tpsL t* ! *j2*) [**0**, **1**] = (⌊*binencode* (*take* (*Suc t*) *zs*)⌋, *Suc* (*2* ∗ *t*) + *2*)
      by *simp*
    moreover have *tpsL1 t* = (*tpsL t*)[*j2* := (⌊*binencode* (*take* (*Suc t*) *zs*)⌋, *Suc* (*2* ∗ *t*) + *2*)]
      unfolding *tpsL1-def tpsL-def* **by** (*simp only*: *list-update-overwrite*)
    ultimately show *tpsL1 t* = (*tpsL t*)[*j2* := *inscribe* (*tpsL t* ! *j2*) [**0**, **1**]]
      using *that* **by** *simp*
  qed
qed

lemma *tm3* [*transforms-intros*]:
  assumes *t* < *length zs*
  shows *transforms tm3* (*tpsL t*) *6* (*tpsL1 t*)
  unfolding *tm3-def*
proof (*tform tps*: *jk assms tpsL-def*)
  have *1*: *tpsL t* ! *j2* = (⌊*binencode* (*take t zs*)⌋, *Suc* (*2* ∗ *t*))
    using *tpsL-def jk* **by** *simp*
  have *2*: *length* (*binencode* (*take t zs*)) = *2* ∗ *t*
    using *assms* **by** *simp*
  show *read* (*tpsL t*) ! *j1* ≠ **0** ⟹ *zs* ! *t* = | ∨ *zs* ! *t* = ♯ ∨ *zs* ! *t* = **1**
    using *assms zs read-tpsL* **by** *auto*
  show *tpsL1 t* = (*tpsL t*)[*j2* := *inscribe* (*tpsL t* ! *j2*) [**0**, **0**]] **if** *read* (*tpsL t*) ! *j1* = **0**
  proof −
    have ∗: *zs* ! *t* = **0**
      using *read-tpsL*[*OF assms*] *that* **by** *simp*
    have *inscribe* (*tpsL t* ! *j2*) [**0**, **0**] = (⌊*binencode* (*take t zs*) @ [**0**, **0**]⌋, *Suc* (*2* ∗ *t*) + *2*)
      using *inscribe-contents 1 2*
      by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-eq-plus1 add-Suc-shift list.size*(*3*) *list.size*(*4*))
    moreover have *binencode* (*take t zs*) @ [**0**, **0**] = *binencode* (*take* (*Suc t*) *zs*)
      using ∗ *assms binencode-snoc* **by** (*metis take-Suc-conv-app-nth*)
    ultimately have *inscribe* (*tpsL t* ! *j2*) [**0**, **0**] = (⌊*binencode* (*take* (*Suc t*) *zs*)⌋, *Suc* (*2* ∗ *t*) + *2*)
      by *simp*
    moreover have *tpsL1 t* = (*tpsL t*)[*j2* := (⌊*binencode* (*take* (*Suc t*) *zs*)⌋, *Suc* (*2* ∗ *t*) + *2*)]
      unfolding *tpsL1-def tpsL-def* **by** (*simp only*: *list-update-overwrite*)

ultimately show *tpsL1 t* = (*tpsL t*)[*j2* := *inscribe* (*tpsL t* ! *j2*) [**0**, **0**]]
using *that* **by** *simp*
qed
qed

lemma *tm4* [*transforms-intros*]:
assumes *t* < *length zs*
shows *transforms tm4* (*tpsL t*) *7* (*tpsL* (*Suc t*))
unfolding *tm4-def*
proof (*tform tps*: *assms tpsL1-def jk*)
have ∗: *tpsL1 t* ! *j1* = (⌊*zs*⌋, *Suc t*)
using *tpsL1-def jk* **by** *simp*
show *tpsL* (*Suc t*) = (*tpsL1 t*)[*j1* := *tpsL1 t* ! *j1* |+| *1*]
using *tpsL-def tpsL1-def* **using** *jk* ∗ **by** (*auto simp add*: *list-update-swap*[*of - j1*])
qed

lemma *tm5*:
assumes *ttt* = *9* ∗ *length zs* + *1*
shows *transforms tm5* (*tpsL 0*) *ttt* (*tpsL* (*length zs*))
unfolding *tm5-def*
proof (*tform*)
show ⋀*t*. *t* < *length zs* ⟹ *read* (*tpsL t*) ! *j1* ≠ □
using *read-tpsL zs* **by** *auto*
show ¬ *read* (*tpsL* (*length zs*)) ! *j1* ≠ □
proof −
have *tpsL* (*length zs*) ! *j1* = (⌊*zs*⌋, *Suc* (*length zs*))
using *tpsL-def jk* **by** *simp*
moreover have *j1* < *length* (*tpsL* (*length zs*))
using *tpsL-def jk* **by** *simp*
ultimately have *read* (*tpsL* (*length zs*)) ! *j1* = *tape-read* (⌊*zs*⌋, *Suc* (*length zs*))
using *tapes-at-read′* **by** *fastforce*
also have ... = □
using *contents-outofbounds* **by** *simp*
finally show *?thesis*
by *simp*
qed
show *length zs* ∗ (*7* + *2*) + *1* ≤ *ttt*
using *assms* **by** *simp*
qed

lemma *tpsL*: *tpsL* (*length zs*) = *tps0*
[*j1* := (⌊*zs*⌋, *Suc* (*length zs*)),
j2 := (⌊*binencode zs*⌋, *Suc* (*2* ∗ (*length zs*)))]
unfolding *tpsL-def* **using** *tps0 jk* **by** *simp*

lemma *tm5′*:
assumes *ttt* = *9* ∗ *length zs* + *1*
shows *transforms tm5 tps0 ttt* (*tpsL* (*length zs*))
using *assms tm5 tpsL-0* **by** *simp*

end

end

lemma *transforms-tm-binencodeI* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx*
fixes *tps tps′* :: *tape list* **and** *ttt k* :: *nat* **and** *zs* :: *symbol list*
assumes *k* = *length tps j1* ≠ *j2 0* < *j2 j1* < *k j2* < *k*
and *binencodable zs*
assumes
tps ! *j1* = (⌊*zs*⌋, *1*)
tps ! *j2* = (⌊[]⌋, *1*)
assumes *ttt* = *9* ∗ *length zs* + *1*

assumes *tps′* ≡ *tps*
[*j1* := (⌊*zs*⌋, *Suc* (*length zs*)),
j2 := (⌊*binencode zs*⌋, *Suc* (*2* ∗ (*length zs*)))]
shows *transforms* (*tm-binencode j1 j2*) *tps ttt tps′*
proof −
interpret *loc*: *turing-machine-binencode j1 j2* **.**
show *?thesis*
using *assms loc.tm5′ loc.tm5-eq-tm-binencode loc.tpsL* **by** *simp*
qed

The next command reads chunks of two symbols over **01** from one tape and writes to another tape the corresponding symbol over **01**|♯. The first symbol of each chunk is memorized on the last tape. If the end of the input (that is, a blank symbol) is encountered, the command stops without writing another symbol.

definition *cmd-bindec* :: *tapeidx* ⇒ *tapeidx* ⇒ *command* **where**
cmd-bindec j1 j2 rs ≡
if rs ! *j1* = *0 then* (*1*, *map* (λ*z*. (*z*, *Stay*)) *rs*)
else (*0*,
map (λ*i*.
if last rs = ▷
then if i = *j1 then* (*rs* ! *i*, *Right*)
else if i = *j2 then* (*rs* ! *i*, *Stay*)
else if i = *length rs* − *1 then* (*tosym* (*todigit* (*rs* ! *j1*)), *Stay*)
else (*rs* ! *i*, *Stay*)
else if i = *j1 then* (*rs* ! *i*, *Right*)
else if i = *j2 then* (*decsym* (*last rs*) (*rs* ! *j1*), *Right*)
else if i = *length rs* − *1 then* (*1*, *Stay*)
else (*rs* ! *i*, *Stay*))
[*0*..<*length rs*])

The Turing machine performing the decoding:

definition *tm-bindec* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-bindec j1 j2 = [*cmd-bindec j1 j2*]

context
fixes *j1 j2* :: *tapeidx* **and** *k* :: *nat*
assumes *j-less*: *j1* < *k j2* < *k*
and *j-gt*: *0* < *j2*
begin

lemma *turing-command-bindec*:
assumes *G* ≥ *6*
shows *turing-command* (*Suc k*) *1 G* (*cmd-bindec j1 j2*)
proof
show ⋀*gs*. *length gs* = *Suc k* ⟹ *length* ([!!] *cmd-bindec j1 j2 gs*) = *length gs*
using *cmd-bindec-def* **by** *simp*
show *cmd-bindec j1 j2 gs* [.] *j* < *G*
if *length gs* = *Suc k* ⋀*i*. *i* < *length gs* ⟹ *gs* ! *i* < *G j* < *length gs*
for *gs j*
proof (*cases gs* ! *j1* = □)
case *True*
then show *?thesis*
using *that cmd-bindec-def* **by** *simp*
next
case *else*: *False*
show *?thesis*
proof (*cases last gs* = ▷)
case *True*
then have *snd* (*cmd-bindec j1 j2 gs*) = *map* (λ*i*.
if i = *j1 then* (*gs* ! *i*, *Right*)
else if i = *j2 then* (*gs* ! *i*, *Stay*)
else if i = *length gs* − *1 then* (*todigit* (*gs* ! *j1*) + *2*, *Stay*)
else (*gs* ! *i*, *Stay*)) [*0*..<*length gs*]

using *cmd-bindec-def else* **by** *simp*
then have *cmd-bindec j1 j2 gs* [!] $j =$
($if\ j = j1\ then\ (gs\ !\ j,\ Right)$
$else\ if\ j = j2\ then\ (gs\ !\ j,\ Stay)$
$else\ if\ j = length\ gs - 1\ then\ (todigit\ (gs\ !\ j1) + 2,\ Stay)$
$else\ (gs\ !\ j,\ Stay))$
using *that*(*3*) **by** *simp*
then have *cmd-bindec j1 j2 gs* [.] $j =$
$(if\ j = j1\ then\ gs\ !\ j$
$else\ if\ j = j2\ then\ gs\ !\ j$
$else\ if\ j = length\ gs - 1\ then\ todigit\ (gs\ !\ j1) + 2$
$else\ gs\ !\ j)$
by *simp*
then show *?thesis*
using *that assms* **by** *simp*
next
case *False*
then have $snd\ (cmd\text{-}bindec\ j1\ j2\ gs) = map\ (\lambda i.$
$if\ i = j1\ then\ (gs\ !\ i,\ Right)$
$else\ if\ i = j2\ then\ (2 * todigit\ (last\ gs) + todigit\ (gs\ !\ j1) + 2,\ Right)$
$else\ if\ i = length\ gs - 1\ then\ (1,\ Stay)$
$else\ (gs\ !\ i,\ Stay))\ [0..<length\ gs]$
using *cmd-bindec-def else* **by** *simp*
then have *cmd-bindec j1 j2 gs* [!] $j =$
$(if\ j = j1\ then\ (gs\ !\ j,\ Right)$
$else\ if\ j = j2\ then\ (2 * todigit\ (last\ gs) + todigit\ (gs\ !\ j1) + 2,\ Right)$
$else\ if\ j = length\ gs - 1\ then\ (1,\ Stay)$
$else\ (gs\ !\ j,\ Stay))$
using *that*(*3*) **by** *simp*
then have *cmd-bindec j1 j2 gs* [.] $j =$
$(if\ j = j1\ then\ gs\ !\ j$
$else\ if\ j = j2\ then\ 2 * todigit\ (last\ gs) + todigit\ (gs\ !\ j1) + 2$
$else\ if\ j = length\ gs - 1\ then\ 1$
$else\ gs\ !\ j)$
by *simp*
moreover have $last\ gs < G$
using *that assms*
by (*metis add-lessD1 diff-less last-conv-nth length-greater-0-conv less-numeral-extra*(*1*)
less-numeral-extra(*4*) *list.size*(*3*) *plus-1-eq-Suc*)
ultimately show *?thesis*
using *that assms* **by** *simp*
qed
qed
show *cmd-bindec j1 j2 gs* [.] $0 = gs\ !\ 0$ **if** $length\ gs = Suc\ k\ 0 < Suc\ k$ **for** *gs*
proof ($cases\ last\ gs = 1$)
case *True*
moreover have $0 < length\ gs\ 0 \neq length\ gs - 1$
using *that j-less* **by** *simp-all*
ultimately show *?thesis*
using *cmd-bindec-def* **by** *simp*
next
case *False*
moreover have $0 < length\ gs\ 0 \neq j2\ 0 \neq length\ gs - 1$
using *that j-gt j-less* **by** *simp-all*
ultimately show *?thesis*
using *cmd-bindec-def* **by** *simp*
qed
show $\bigwedge gs.\ length\ gs = Suc\ k \Longrightarrow [*]\ (cmd\text{-}bindec\ j1\ j2\ gs) \le 1$
using *cmd-bindec-def* **by** *simp*
qed

lemma *tm-bindec-tm*: $G \ge 6 \Longrightarrow turing\text{-}machine\ (Suc\ k)\ G\ (tm\text{-}bindec\ j1\ j2)$
unfolding *tm-bindec-def* **using** *j-less*(*2*) *j-gt turing-command-bindec cmd-bindec-def* **by** *auto*

context
  fixes *tps* :: *tape list* **and** *zs* :: *symbol list*
  assumes *j1-neq*: $j1 \neq j2$
    and *lentps*: *Suc k* = *length tps*
    and *bs*: *bit-symbols zs*
begin

lemma *sem-cmd-bindec-gt*:
  assumes *tps* ! *j1* = (⌊*zs*⌋, *i*)
    and *i* > *length zs*
  shows *sem* (*cmd-bindec j1 j2*) (*0*, *tps*) = (*1*, *tps*)
proof (*rule semI*)
  show *proper-command* (*Suc k*) (*cmd-bindec j1 j2*)
    using *cmd-bindec-def* **by** *simp*
  show *length tps* = *Suc k*
    using *lentps* **by** *simp*
  show *length tps* = *Suc k*
    using *lentps* **by** *simp*
  have *read tps* ! *j1* = □
    using *assms* **by** (*metis contents-outofbounds fst-conv j-less*(*1*) *lentps less-Suc-eq snd-conv tapes-at-read′*)
  moreover from *this* **show** *fst* (*cmd-bindec j1 j2* (*read tps*)) = *1*
    by (*simp add*: *cmd-bindec-def*)
  ultimately show ⋀*j*. *j* < *Suc k* ⟹ *act* (*cmd-bindec j1 j2* (*read tps*) [!] *j*) (*tps* ! *j*) = *tps* ! *j*
    using *assms cmd-bindec-def act-Stay lentps read-length* **by** *simp*
qed

lemma *sem-cmd-bindec-1*:
  assumes *tps* ! *k* = ⌈▷⌉
    and *tps* ! *j1* = (⌊*zs*⌋, *i*)
    and *i* > *0*
    and *i* ≤ *length zs*
    and *tps′* = *tps* [*j1* := *tps* ! *j1* |+| *1*, *k* := ⌈*todigit* (*tps* :.: *j1*) + *2*⌉]
  shows *sem* (*cmd-bindec j1 j2*) (*0*, *tps*) = (*0*, *tps′*)
proof (*rule semI*)
  show *proper-command* (*Suc k*) (*cmd-bindec j1 j2*)
    using *cmd-bindec-def* **by** *simp*
  show *length tps* = *Suc k*
    using *lentps* **by** *simp*
  show *length tps′* = *Suc k*
    using *lentps assms*(*5*) **by** *simp*
  have *read*: *read tps* ! *j1* ≠ □
    using *assms*(*2*,*3*,*4*) *bs j-less*(*1*) *tapes-at-read′*[*of j1 tps*] *contents-inbounds*[*OF assms*(*3*,*4*)] *lentps*
      proper-symbols-ne0[*OF proper-bit-symbols*[*OF bs*]]
    by (*metis One-nat-def Suc-less-eq Suc-pred fst-conv le-imp-less-Suc less-SucI snd-eqD*)
  then show *fst* (*cmd-bindec j1 j2* (*read tps*)) = *0*
    using *cmd-bindec-def* **by** *simp*
  show *act* (*cmd-bindec j1 j2* (*read tps*) [!] *j*) (*tps* ! *j*) = *tps′* ! *j*
    if *j* < *Suc k* **for** *j*
  proof −
    let *?rs* = *read tps*
    have *last ?rs* = *1*
      using *assms*(*1*) *lentps onesie-read read-length tapes-at-read′*
      by (*metis* (*mono-tags*, *lifting*) *last-length lessI*)
    then have ∗: *snd* (*cmd-bindec j1 j2 ?rs*) = *map* (λ*i*.
        if i = *j1 then* (*?rs* ! *i*, *Right*)
        else if i = *j2 then* (*?rs* ! *i*, *Stay*)
        else if i = *length ?rs* − *1 then* (*if ?rs* ! *j1* = **1** *then* **1** *else* **0**, *Stay*)
        else (*?rs* ! *i*, *Stay*)) [*0*..<*length ?rs*]
      using *read cmd-bindec-def* **by** *simp*
    have *length ?rs* = *Suc k*
      by (*simp add*: *lentps read-length*)
    then have *len*: *j* < *length ?rs*

```
      using that by simp
    have k: k = length ?rs − 1
      by (simp add: ‹length ?rs = Suc k›)
    consider j = j1 | j ≠ j1 ∧ j = j2 | j ≠ j1 ∧ j ≠ j2 ∧ j = k | j ≠ j1 ∧ j ≠ j2 ∧ j ≠ k
      by auto
    then show ?thesis
    proof (cases)
      case 1
      then have cmd-bindec j1 j2 ?rs [!] j = (?rs ! j1, Right)
        using ∗ len by simp
      then show ?thesis
        using act-Right 1 assms(5) j-less(1) lentps by simp
    next
      case 2
      then have cmd-bindec j1 j2 ?rs [!] j = (?rs ! j2, Stay)
        using ∗ len by simp
      then show ?thesis
        using 2 act-Stay assms(5) j-less(2) lentps by simp
    next
      case 3
      then have cmd-bindec j1 j2 ?rs [!] j = (todigit (?rs ! j1) + 2, Stay)
        using k ∗ len by simp
      then show ?thesis
        using 3 assms(1,5) act-onesie j-less(1) lentps tapes-at-read'
        by (metis length-list-update less-Suc-eq nth-list-update)
    next
      case 4
      then have cmd-bindec j1 j2 ?rs [!] j = (?rs ! j, Stay)
        using k ∗ len that by simp
      then show ?thesis
        using 4 act-Stay assms(5) lentps that by simp
    qed
  qed
qed

lemma sem-cmd-bindec-23:
  assumes tps ! k = ⌈s⌉
    and s = 0 ∨ s = 1
    and tps ! j1 = (⌊zs⌋, i)
    and i > 0
    and i ≤ length zs
    and tps' = tps
      [j1 := tps ! j1 |+| 1,
       j2 := tps ! j2 |:=| decsym s (tps :.: j1) |+| 1,
       k := ⌈▷⌉]
  shows sem (cmd-bindec j1 j2) (0, tps) = (0, tps')
proof (rule semI)
  show proper-command (Suc k) (cmd-bindec j1 j2)
    using cmd-bindec-def by simp
  show length tps = Suc k
    using lentps by simp
  show length tps' = Suc k
    using lentps assms(6) by simp
  have read: read tps ! j1 ≠ □
    using assms(3−5) bs tapes-at-read'[of j1 tps] contents-inbounds[OF assms(4,5)] lentps
    by (metis One-nat-def Suc-less-eq Suc-pred fst-conv j-less(1) le-imp-less-Suc
      less-imp-le-nat not-one-less-zero proper-bit-symbols snd-conv)
  show fst (cmd-bindec j1 j2 (read tps)) = 0
    using read cmd-bindec-def by simp
  show act (cmd-bindec j1 j2 (read tps) [!] j) (tps ! j) = tps' ! j
    if j < Suc k for j
  proof −
    let ?rs = read tps
```

```
    have last ?rs ≠ 1
      using assms(1,2) lentps onesie-read read-length tapes-at-read'
      by (metis Suc-1 diff-Suc-1 last-conv-nth lessI list.size(3) n-not-Suc-n numeral-One
        numeral-eq-iff old.nat.distinct(1) semiring-norm(86))
    then have ∗: snd (cmd-bindec j1 j2 ?rs) = map (λi.
        if i = j1 then (?rs ! i, Right)
        else if i = j2 then (2 ∗ todigit (last ?rs) + todigit (?rs ! j1) + 2, Right)
        else if i = length ?rs − 1 then (1, Stay)
        else (?rs ! i, Stay)) [0..<length ?rs]
      using read cmd-bindec-def by simp
    have lenrs: length ?rs = Suc k
      by (simp add: lentps read-length)
    then have len: j < length ?rs
      using that by simp
    have k: k = length ?rs − 1
      by (simp add: lenrs)
    consider j = j1 | j ≠ j1 ∧ j = j2 | j ≠ j1 ∧ j ≠ j2 ∧ j = k | j ≠ j1 ∧ j ≠ j2 ∧ j ≠ k
      by auto
    then show ?thesis
    proof (cases)
      case 1
      then show ?thesis
        using ∗ len act-Right assms(6) j-less(1) j1-neq lentps by simp
    next
      case 2
      then have cmd-bindec j1 j2 ?rs [!] j = (2 ∗ todigit (last ?rs) + todigit (?rs ! j1) + 2, Right)
        using ∗ len by simp
      moreover have last ?rs = s
        using assms(1,2) lenrs k onesie-read tapes-at-read'
        by (metis last-conv-nth length-0-conv lentps lessI old.nat.distinct(1))
      moreover have ?rs ! j1 = tps :.: j1
        using j-less(1) lentps tapes-at-read' by simp
      ultimately show ?thesis
        using 2 assms(6) act-Right' j-less lentps by simp
    next
      case 3
      then show ?thesis
        using ∗ len k act-onesie assms(1,6) lentps by simp
    next
      case 4
      then have cmd-bindec j1 j2 ?rs [!] j = (?rs ! j, Stay)
        using k ∗ len by simp
      then show ?thesis
        using 4 act-Stay assms(6) lentps that by simp
    qed
  qed
qed

end

lemma transits-tm-bindec:
  fixes tps :: tape list and zs :: symbol list
  assumes j1-neq: j1 ≠ j2
    and j1j2: 0 < j2 j1 < k j2 < k
    and lentps: Suc k = length tps
    and bs: bit-symbols zs
  assumes tps ! k = ⌈▷⌉
    and tps ! j1 = (⌊zs⌋, 2 ∗ i + 1)
    and tps ! j2 = (⌊bindecode (take (2 ∗ i) zs)⌋, Suc i)
    and i < length zs div 2
    and tps' = tps
      [j1 := (⌊zs⌋, 2 ∗ (Suc i) + 1),
       j2 := (⌊bindecode (take (2 ∗ Suc i) zs)⌋, Suc (Suc i))]
```

shows *transits* (*tm-bindec j1 j2*) (*0*, *tps*) *2* (*0*, *tps'*)
proof −
define *tps1* **where** *tps1* = *tps*
[*j1* := (⌊*zs*⌋, *2* ∗ *i* + *2*),
k := ⌈*todigit* (*tps* :.: *j1*) + *2*⌉]
let *?i* = *2* ∗ *i* + *1*
let *?M* = *tm-bindec j1 j2*
have *ilen*: *?i* < *length zs*
using *assms*(*10*) **by** *simp*
have *exe ?M* (*0*, *tps*) = *sem* (*cmd-bindec j1 j2*) (*0*, *tps*)
using *tm-bindec-def exe-lt-length* **by** *simp*
also have ... =
(*if ?i* ≤ *length zs then 0 else 1*,
tps[*j1* := *tps* ! *j1* |+| *1*, *k* := ⌈*todigit* (*tps* :.: *j1*) + *2*⌉])
using *ilen bs assms*(*7*,*8*) *sem-cmd-bindec-1 j1-neq lentps* **by** *simp*
also have ... = (*0*, *tps*[*j1* := *tps* ! *j1* |+| *1*, *k* := ⌈*todigit* (*tps* :.: *j1*) + *2*⌉])
using *ilen* **by** *simp*
also have ... = (*0*, *tps1*)
using *tps1-def assms* **by** *simp*
finally have *step1*: *exe ?M* (*0*, *tps*) = (*0*, *tps1*) **.**

let *?s* = *tps1* :.: *k*
have *tps* :.: *j1* = *zs* ! (*2* ∗ *i*)
using *assms*(*8*) *ilen* **by** *simp*
then have *?s* = *todigit* (*zs* ! (*2* ∗ *i*)) + *2*
using *tps1-def lentps* **by** *simp*
then have *?s* = *zs* ! (*2* ∗ *i*)
using *ilen bs* **by** (*smt* (*verit*) *One-nat-def Suc-1 add-2-eq-Suc' add-lessD1 numeral-3-eq-3*)
moreover have *tps1* :.: *j1* = *zs* ! *?i*
using *tps1-def ilen lentps j1j2* **by** *simp*
ultimately have ∗: *decsym ?s* (*tps1* :.: *j1*) = *decsym* (*zs* ! (*2*∗*i*)) (*zs* ! (*Suc* (*2*∗*i*)))
by *simp*

have *exe ?M* (*0*, *tps1*) = *sem* (*cmd-bindec j1 j2*) (*0*, *tps1*)
using *tm-bindec-def exe-lt-length* **by** *simp*
also have ... =
(*if Suc ?i* ≤ *length zs then 0 else 1*,
tps1[*j1* := *tps1* ! *j1* |+| *1*, *j2* := *tps1* ! *j2* |:=| *2* ∗ *todigit ?s* + *todigit* (*tps1* :.: *j1*) + *2* |+| *1*, *k* := ⌈▷⌉])
proof −
have *1*: *tps1* ! *k* = ⌈*?s*⌉
using *tps1-def lentps* **by** *simp*
have *2*: *?s* = *2* ∨ *?s* = *3*
using *tps1-def lentps* **by** *simp*
have *3*: *tps1* ! *j1* = (⌊*zs*⌋, *Suc ?i*)
using *tps1-def lentps Suc-1 add-Suc-right j-less*(*1*) *less-Suc-eq nat-neq-iff nth-list-update-eq nth-list-update-neq*
by *simp*
have *4*: *Suc ?i* > *0*
by *simp*
have *5*: *Suc k* = *length tps1*
by (*simp add*: *lentps tps1-def*)
show *?thesis*
using *ilen sem-cmd-bindec-23*[*of tps1 zs ?s Suc ?i*, *OF j1-neq 5 bs 1 2 3 4*] **by** *simp*
qed
also have ... =
(*0*, *tps1*[*j1* := *tps1* ! *j1* |+| *1*, *j2* := *tps1* ! *j2* |:=| *decsym ?s* (*tps1* :.: *j1*) |+| *1*, *k* := ⌈▷⌉])
using *assms*(*10*) *length-binencode* **by** *simp*
also have ... =
(*0*, *tps1*[*j1* := *tps1* ! *j1* |+| *1*, *j2* := *tps1* ! *j2* |:=| *decsym* (*zs* ! (*2*∗*i*)) (*zs* ! (*Suc* (*2*∗*i*))) |+| *1*, *k* := ⌈▷⌉])
(**is** - = (*0*, *?tps*))
using ∗ **by** *simp*
also have ... = (*0*, *tps'*)
proof −
have *len'*: *length tps'* = *Suc k*

using *assms lentps* **by** *simp*
have *len1*: *length tps1* = *Suc k*
using *assms lentps tps1-def* **by** *simp*
have *1*: *?tps* ! *k* = *tps′* ! *k*
using *assms*(*7*,*11*) **by** (*simp add*: *j-less*(*1*) *j-less*(*2*) *len1 nat-neq-iff*)
have *2*: *?tps* ! *j1* = *tps′* ! *j1*
using *assms*(*11*) *j1-neq j-less*(*1*) *lentps tps1-def* **by** *simp*
have *?tps* ! *j2* = *tps1* ! *j2* |:=| *decsym* (*zs* ! (*2*∗*i*)) (*zs* ! (*Suc* (*2*∗*i*))) |+| *1*
by (*simp add*: *j-less*(*2*) *len1 less-Suc-eq nat-neq-iff*)
also have ... = *tps* ! *j2* |:=| *decsym* (*zs* ! (*2*∗*i*)) (*zs* ! (*Suc* (*2*∗*i*))) |+| *1*
using *tps1-def j1-neq j-less*(*2*) *len1* **by** *force*
also have ... = (⌊*bindecode* (*take* (*2* ∗ *i*) *zs*)⌋, *Suc i*) |:=| *decsym* (*zs* ! (*2*∗*i*)) (*zs* ! (*Suc* (*2*∗*i*))) |+| *1*
using *assms*(*9*) *j1-neq j-less*(*2*) *len1 tps1-def* **by** *simp*
also have ... = (⌊*bindecode* (*take* (*2* ∗ *i*) *zs*)⌋(*Suc i* := *decsym* (*zs* ! (*2*∗*i*)) (*zs* ! (*Suc* (*2*∗*i*)))), *Suc* (*Suc i*))
by *simp*
also have ... = (⌊*bindecode* (*take* (*2* ∗ *i*) *zs*) @ [*decsym* (*zs* ! (*2*∗*i*)) (*zs* ! (*Suc* (*2*∗*i*)))]⌋, *Suc* (*Suc i*))
using *contents-snoc*[*of bindecode* (*take* (*2* ∗ *i*) *zs*)] *ilen length-bindecode*
proof −
have *length* (*bindecode* (*take* (*2* ∗ *i*) *zs*)) = *i*
using *ilen length-bindecode* **by** *simp*
then show *?thesis*
using *contents-snoc*[*of bindecode* (*take* (*2* ∗ *i*) *zs*)] **by** *simp*
qed
also have ... = (⌊*bindecode* (*take* (*2* ∗ *Suc i*) *zs*)⌋, *Suc* (*Suc i*))
using *bindecode-take-snoc ilen* **by** *simp*
also have ∗: ... = *tps′* ! *j2*
by (*metis assms*(*11*) *j-less*(*2*) *length-list-update lentps less-Suc-eq nth-list-update-eq*)
finally have *?tps* ! *j2* = *tps′* ! *j2* **.**
with *1 2 assms*(*11*) ∗ **show** *?thesis*
unfolding *tps1-def*
by (*smt* (*verit*) *j-less*(*1*) *j-less*(*2*) *lentps less-Suc-eq list-update-id list-update-overwrite list-update-swap nat-neq-iff nth-list-update-eq nth-list-update-neq*)
qed
finally have *exe ?M* (*0*, *tps1*) = (*0*, *tps′*) **.**
then have *execute ?M* (*0*, *tps*) *2* = (*0*, *tps′*)
using *step1* **by** (*simp add*: *numeral-2-eq-2*)
then show *transits* (*tm-bindec j1 j2*) (*0*, *tps*) *2* (*0*, *tps′*)
using *execute-imp-transits* **by** *blast*
qed

context
fixes *tps* :: *tape list* **and** *zs* :: *symbol list*
assumes *j1-neq*: *j1* ≠ *j2*
and *j1j2*: *0* < *j2 j1* < *k j2* < *k*
and *lentps*: *Suc k* = *length tps*
and *bs*: *bit-symbols zs*
begin

lemma *transits-tm-bindec′*:
assumes *tps* ! *k* = ⌈▷⌉
and *tps* ! *j1* = (⌊*zs*⌋, *1*)
and *tps* ! *j2* = (⌊[]⌋, *1*)
and *i* ≤ *length zs div 2*
and *tps′* = *tps*
[*j1* := (⌊*zs*⌋, *2* ∗ *i* + *1*),
j2 := (⌊*bindecode* (*take* (*2* ∗ *i*) *zs*)⌋, *Suc i*)]
shows *transits* (*tm-bindec j1 j2*) (*0*, *tps*) (*2* ∗ *i*) (*0*, *tps′*)
using *assms*(*4*,*5*)
proof (*induction i arbitrary*: *tps′*)
case *0*
then show *?case*
using *assms*(*2*,*3*) **by** (*metis One-nat-def add.commute div-mult-self1-is-m execute.simps*(*1*)
le-numeral-extra(*3*) *length-bindecode length-greater-0-conv list.size*(*3*) *list-update-id*

mult-0-right plus-1-eq-Suc take0 transits-def zero-less-numeral)
next
  case (*Suc i*)
  define *tpsi* **where** *tpsi* = *tps*
    [*j1* := (⌊*zs*⌋, *2* ∗ *i* + *1*),
     j2 := (⌊*bindecode* (*take* (*2*∗*i*) *zs*)⌋, *Suc i*)]
  then have *transits* (*tm-bindec j1 j2*) (*0*, *tps*) (*2* ∗ *i*) (*0*, *tpsi*)
    using *Suc* **by** *simp*
  moreover have *transits* (*tm-bindec j1 j2*) (*0*, *tpsi*) *2* (*0*, *tps*′)
  proof −
    have *1*: *tpsi* ! *k* = ⌈▷⌉
      using *tpsi-def* **by** (*simp add*: *assms*(*1*) *j-less*(*1*) *j-less*(*2*) *less-not-refl3*)
    have *2*: *tpsi* ! *j1* = (⌊*zs*⌋, *2* ∗ *i* + *1*)
      using *tpsi-def* **by** (*metis j1-neq j-less*(*1*) *lentps less-Suc-eq nth-list-update-eq nth-list-update-neq*)
    have *3*: *tpsi* ! *j2* = (⌊*bindecode* (*take* (*2* ∗ *i*) *zs*)⌋, *Suc i*)
      using *tpsi-def* **by** (*metis j-less*(*2*) *length-list-update lentps less-Suc-eq nth-list-update-eq*)
    have *4*: *i* < *length zs div 2*
      using *Suc* **by** *simp*
    have *5*: *tps*′ = *tpsi*
      [*j1* := (⌊*zs*⌋, *2* ∗ (*Suc i*) + *1*),
       j2 := (⌊*bindecode* (*take* (*2* ∗ *Suc i*) *zs*)⌋, *Suc* (*Suc i*))]
      using *Suc tpsi-def* **by** (*metis* (*no-types*, *opaque-lifting*) *list-update-overwrite list-update-swap*)
    have *6*: *Suc k* = *length tpsi*
      using *tpsi-def lentps* **by** *simp*
    show *?thesis*
      using *transits-tm-bindec*[*OF j1-neq j1j2 6 bs 1 2 3 4 5*] **.**
  qed
  ultimately show *transits* (*tm-bindec j1 j2*) (*0*, *tps*) (*2* ∗ (*Suc i*)) (*0*, *tps*′)
    using *transits-additive* **by** *fastforce*
qed

corollary *transits-tm-bindec*′′:
  assumes *tps* ! *k* = ⌈▷⌉
    and *tps* ! *j1* = (⌊*zs*⌋, *1*)
    and *tps* ! *j2* = (⌊[]⌋, *1*)
    and *l* = *length zs div 2*
    and *tps*′ = *tps*
      [*j1* := (⌊*zs*⌋, *2* ∗ *l* + *1*),
       j2 := (⌊*bindecode* (*take* (*2* ∗ *l*) *zs*)⌋, *Suc l*)]
  shows *transits* (*tm-bindec j1 j2*) (*0*, *tps*) (*2* ∗ *l*) (*0*, *tps*′)
  using *assms transits-tm-bindec*′ **by** *simp*

In case the input is of odd length, that is, malformed:

lemma *transforms-tm-bindec-odd*:
  assumes *tps* ! *k* = ⌈▷⌉
    and *tps* ! *j1* = (⌊*zs*⌋, *1*)
    and *tps* ! *j2* = (⌊[]⌋, *1*)
    and *tps*′ = *tps*
      [*j1* := (⌊*zs*⌋, *2* ∗ *l* + *2*),
       j2 := (⌊*bindecode zs*⌋, *Suc l*),
       k := ⌈*todigit* (*last zs*) + *2*⌉]
    and *l* = *length zs div 2*
    and *Suc* (*2* ∗ *l*) = *length zs*
  shows *transforms* (*tm-bindec j1 j2*) *tps* (*2* ∗ *l* + *2*) *tps*′
proof −
  let *?ys* = *bindecode* (*take* (*2* ∗ *l*) *zs*)
  let *?i* = *2* ∗ *l* + *1*
  let *?M* = *tm-bindec j1 j2*
  have *ys*: *?ys* = *bindecode zs*
    using *bindecode-odd assms*(*6*) **by** (*metis Suc-eq-plus1*)
  have *zs* ≠ []
    using *assms*(*6*) **by** *auto*
  define *tps1* **where** *tps1* = *tps*

```
    [j1 := (⌊zs⌋, 2 * l + 1),
     j2 := (⌊?ys⌋, Suc l)]
define tps2 where tps2 = tps
  [j1 := (⌊zs⌋, 2 * l + 2),
   j2 := (⌊bindecode zs⌋, Suc l),
   k := ⌈todigit (tps1 :.: j1) + 2⌉]
have transits ?M (0, tps) (2 * l) (0, tps1)
  using tps1-def assms transits-tm-bindec'' by simp
moreover have execute ?M (0, tps1) 1 = (0, tps2)
proof -
  have execute ?M (0, tps1) 1 = exe ?M (0, tps1)
    by simp
  also have ... = sem (cmd-bindec j1 j2) (0, tps1)
    using exe-lt-length tm-bindec-def by simp
  also have ... = (0, tps1[j1 := tps1 ! j1 |+| 1, k := ⌈todigit (tps1 :.: j1) + 2⌉])
    (is - = (0, ?tps))
  proof -
    have tps1 ! j1 = (⌊zs⌋, ?i)
      using lentps tps1-def j1-neq j-less by simp
    moreover have ?i > 0
      by simp
    moreover have tps1 ! k = tps ! k
      using tps1-def by (simp add: j-less(1) j-less(2) nat-neq-iff)
    moreover have ?i ≤ length zs
      by (simp add: assms(6))
    ultimately have sem (cmd-bindec j1 j2) (0, tps1) = (0, ?tps)
      using sem-cmd-bindec-1 assms(1,4) bit-symbols-binencode bs j1-neq lentps tps1-def
      by (metis length-list-update)
    then show ?thesis
      by simp
  qed
  also have ... = (0, tps2)
  proof -
    have tps2 ! j1 = ?tps ! j1
      using tps1-def tps2-def j1-neq j-less(1) lentps by simp
    moreover have tps2 ! j2 = ?tps ! j2
      using tps1-def tps2-def j1-neq j-less(2) lentps ys by simp
    ultimately have tps2 = ?tps
      using tps2-def tps1-def j-less(1) lentps
      by (smt (verit) list-update-id list-update-overwrite list-update-swap)
    then show ?thesis
      by simp
  qed
  finally show ?thesis .
qed
ultimately have transits ?M (0, tps) (2 * l + 1) (0, tps2)
  using execute-imp-transits transits-additive by blast
moreover have execute ?M (0, tps2) 1 = (1, tps')
proof -
  have execute ?M (0, tps2) 1 = exe ?M (0, tps2)
    by simp
  also have ... = sem (cmd-bindec j1 j2) (0, tps2)
    using exe-lt-length tm-bindec-def by simp
  also have ... = (1, tps2)
  proof -
    have 2 * l + 2 > length zs
      using assms(5,6) by simp
    moreover have tps2 ! j1 = (⌊zs⌋, 2 * l + 2)
      using tps2-def j1-neq j-less(1) lentps by simp
    ultimately show ?thesis
      using sem-cmd-bindec-gt[of tps2 zs 2 * l + 2]
      by (metis bs j1-neq length-list-update lentps tps2-def)
  qed
```

    moreover have *tps2* = *tps'*
    proof −
      have *tps1* :.: *j1* = *last zs*
        using *tps1-def assms* ‹*zs* ≠ []› *contents-inbounds*
        by (*metis Suc-leI add.commute fst-conv j1-neq j-less*(*1*) *last-conv-nth lentps less-Suc-eq*
          nth-list-update-eq nth-list-update-neq plus-1-eq-Suc snd-conv zero-less-Suc)
      then show *?thesis*
        using *tps2-def assms*(*4*) **by** *simp*
    qed
    ultimately show *?thesis*
      by *simp*
  qed
  ultimately have *transits ?M* (*0*, *tps*) (*2* ∗ *l* + *2*) (*1*, *tps'*)
    using *execute-imp-transits transits-additive* **by** (*smt* (*verit*) *ab-semigroup-add-class.add-ac*(*1*) *nat-1-add-1*)
  then show *transforms* (*tm-bindec j1 j2*) *tps* (*2* ∗ *l* + *2*) *tps'*
    using *transforms-def tm-bindec-def* **by** *simp*
qed

In case the input is of even length, that is, properly encoded:

lemma *transforms-tm-bindec-even*:
  assumes *tps* ! *k* = ⌈▷⌉
    and *tps* ! *j1* = (⌊*zs*⌋, *1*)
    and *tps* ! *j2* = (⌊[]⌋, *1*)
    and *tps'* = *tps*
      [*j1* := (⌊*zs*⌋, *2* ∗ *l* + *1*),
       j2 := (⌊*bindecode zs*⌋, *Suc l*)]
    and *l* = *length zs div 2*
    and *2* ∗ *l* = *length zs*
  shows *transforms* (*tm-bindec j1 j2*) *tps* (*2* ∗ *l* + *1*) *tps'*
proof −
  let *?ys* = *bindecode* (*take* (*2* ∗ *l*) *zs*)
  let *?i* = *2* ∗ *l* + *1*
  let *?M* = *tm-bindec j1 j2*
  have *ys*: *?ys* = *bindecode zs*
    using *assms*(*6*) **by** *simp*
  have *transits ?M* (*0*, *tps*) (*2* ∗ *l*) (*0*, *tps'*)
    using *assms transits-tm-bindec''* **by** *simp*
  moreover have *execute ?M* (*0*, *tps'*) *1* = (*1*, *tps'*)
  proof −
    have *execute ?M* (*0*, *tps'*) *1* = *exe ?M* (*0*, *tps'*)
      using *assms*(*4*) **by** *simp*
    also have ... = *sem* (*cmd-bindec j1 j2*) (*0*, *tps'*)
      using *exe-lt-length tm-bindec-def* **by** *simp*
    also have ... = (*1*, *tps'*)
    proof −
      have *tps'* ! *j1* = (⌊*zs*⌋, *?i*)
        using *lentps assms*(*4*) *j1-neq j-less* **by** *simp*
      moreover have *?i* > *length zs*
        using *assms*(*6*) **by** *simp*
      moreover have *tps'* ! *k* = *tps* ! *k*
        using *assms*(*4*) **by** (*simp add*: *j-less*(*1*) *j-less*(*2*) *nat-neq-iff*)
      ultimately have *sem* (*cmd-bindec j1 j2*) (*0*, *tps'*) = (*1*, *tps'*)
        using *sem-cmd-bindec-gt assms*(*1*,*4*) *bit-symbols-binencode bs j1-neq lentps assms*(*4*)
        by *simp*
      then show *?thesis*
        by *simp*
    qed
    finally show *?thesis* **.**
  qed
  ultimately have *transits ?M* (*0*, *tps*) (*2* ∗ *l* + *1*) (*1*, *tps'*)
    using *execute-imp-transits transits-additive* **by** *blast*
  then show *transforms* (*tm-bindec j1 j2*) *tps* (*2* ∗ *l* + *1*) *tps'*
    using *tm-bindec-def transforms-def* **by** *simp*

```
qed

lemma transforms-tm-bindec:
  assumes tps ! k = ⌈▷⌉
    and tps ! j1 = (⌊zs⌋, 1)
    and tps ! j2 = (⌊[]⌋, 1)
    and tps' = tps
      [j1 := (⌊zs⌋, Suc (length zs)),
       j2 := (⌊bindecode zs⌋, Suc (length zs div 2)),
       k := ⌈if even (length zs) then 1 else (todigit (last zs) + 2)⌉]
  shows transforms (tm-bindec j1 j2) tps (Suc (length zs)) tps'
proof (cases even (length zs))
  case True
  then show ?thesis
    using transforms-tm-bindec-even[OF assms(1−3)] assms(1,4) j-less(1) j-less(2)
    by (smt (verit) Suc-eq-plus1 dvd-mult-div-cancel list-update-id list-update-swap nat-neq-iff)
next
  case False
  then show ?thesis
    using assms(4) transforms-tm-bindec-odd[OF assms(1−3)] by simp
qed

end

end
```

Next we eliminate the memorization tape from *tm-bindec*.

```
lemma transforms-cartesian-bindec:
  assumes G ≥ (6 :: nat)
  assumes j1 ≠ j2
    and j1j2: 0 < j2 j1 < k j2 < k
    and k = length tps
    and bit-symbols zs
  assumes tps ! j1 = (⌊zs⌋, 1)
    and tps ! j2 = (⌊[]⌋, 1)
  assumes t = Suc (length zs)
    and tps' = tps
      [j1 := (⌊zs⌋, Suc (length zs)),
       j2 := (⌊bindecode zs⌋, Suc (length zs div 2))]
  shows transforms (cartesian (tm-bindec j1 j2) 4) tps t tps'
proof (rule cartesian-transforms-onesie)
  show turing-machine (Suc k) G (tm-bindec j1 j2)
    using tm-bindec-tm assms(1) j1j2 by simp
  show immobile (tm-bindec j1 j2) k (Suc k)
  proof (standard+)
    fix q :: nat and rs :: symbol list
    assume q < length (tm-bindec j1 j2) length rs = Suc k
    then have ∗: tm-bindec j1 j2 ! q = cmd-bindec j1 j2
      using tm-bindec-def by simp
    moreover have cmd-bindec j1 j2 rs [~] k = Stay
      using cmd-bindec-def ‹length rs = Suc k› j1j2
      by (smt (verit, best) add-diff-inverse-nat diff-zero length-upt lessI less-nat-zero-code
        nat-neq-iff nth-map nth-upt prod.sel(2))
    ultimately show (tm-bindec j1 j2 ! q) rs [~] k = Stay
      using ∗ by simp
  qed
  show 2 ≤ k
    using j1j2 by linarith
  show (1::nat) < 4
    by simp
  show length tps = k
    using assms(3,6) by simp
  show bounded-write (tm-bindec j1 j2) k 4
```

```
proof -
  { fix q :: nat and rs :: symbol list
    assume q: q < length (tm-bindec j1 j2) and rs: length rs = Suc k
    then have tm-bindec j1 j2 ! q = cmd-bindec j1 j2
      using tm-bindec-def by simp
    have cmd-bindec j1 j2 rs [.] (length rs − 1) < 4 ∨ fst (cmd-bindec j1 j2 rs) = 1
    proof (cases rs ! j1 = 0)
      case True
      then show ?thesis
        using cmd-bindec-def by simp
    next
      case else: False
      show ?thesis
      proof (cases last rs = 1)
        case True
        then have snd (cmd-bindec j1 j2 rs) = map (λi.
                if i = j1 then (rs ! i, Right)
                else if i = j2 then (rs ! i, Stay)
                else if i = length rs − 1 then (todigit (rs ! j1) + 2, Stay)
                else (rs ! i, Stay)) [0..<length rs]
          using else cmd-bindec-def by simp
        then have snd (cmd-bindec j1 j2 rs) ! k = (todigit (rs ! j1) + 2, Stay)
          using rs j1j2
          by (smt (verit) add.left-neutral diff-Suc-1 diff-zero length-upt lessI nat-neq-iff nth-map nth-upt)
        then show ?thesis
          using rs by simp
      next
        case False
        then have snd (cmd-bindec j1 j2 rs) = map (λi.
              if i = j1 then (rs ! i, Right)
              else if i = j2 then (2 ∗ todigit (last rs) + todigit (rs ! j1) + 2, Right)
              else if i = length rs − 1 then (1, Stay)
              else (rs ! i, Stay)) [0..<length rs]
          using else cmd-bindec-def by simp
        then have snd (cmd-bindec j1 j2 rs) ! k = (1, Stay)
          using rs j1j2
          by (smt (verit) add.left-neutral diff-Suc-1 diff-zero length-upt lessI nat-neq-iff nth-map nth-upt)
        then show ?thesis
          using rs by simp
      qed
    qed
  }
  then show ?thesis
    using bounded-write-def tm-bindec-def by simp
qed
let ?c = if even (length zs) then ▷ else (todigit (last zs) + 2)
show transforms (tm-bindec j1 j2) (tps @ [⌈▷⌉]) t (tps' @ [⌈?c⌉])
  (is transforms - ?tps t ?tps')
proof -
  have ?tps ! k = ⌈▷⌉
    by (simp add: assms(6))
  moreover have ?tps ! j1 = (⌊zs⌋, 1)
    by (metis assms(6) assms(8) j1j2(2) nth-append)
  moreover have ?tps ! j2 = (⌊[]⌋, 1)
    by (metis assms(6) assms(9) j1j2(3) nth-append)
  moreover have ?tps' = ?tps
    [j1 := (⌊zs⌋, Suc (length zs)),
     j2 := (⌊bindecode zs⌋, Suc (length zs div 2)),
     k := ⌈?c⌉]
    by (metis (no-types, lifting) assms(6,11) j1j2(2,3) length-list-update list-update-append1 list-update-length)
  ultimately show ?thesis
    using transforms-tm-bindec[of j1 k j2 ?tps zs ?tps'] assms by simp
qed
```

qed

The next Turing machine decodes a bit symbol sequence given on tape j_1 into a quaternary symbol sequence output to tape j_2. It executes the previous TM followed by carriage returns on the tapes j_1 and j_2.

definition *tm-bindecode* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-bindecode j1 j2 ≡ *cartesian* (*tm-bindec j1 j2*) *4* ;; *tm-cr j1* ;; *tm-cr j2*

lemma *tm-bindecode-tm*:
  fixes *j1 j2* :: *tapeidx* **and** *G k* :: *nat*
  assumes *G* ≥ *6* **and** *j1* < *k* **and** *j2* < *k* **and** *0* < *j2* **and** *j1* ≠ *j2*
  shows *turing-machine k G* (*tm-bindecode j1 j2*)
  using *assms tm-bindec-tm tm-bindecode-def cartesian-tm tm-cr-tm* **by** *simp*

locale *turing-machine-bindecode* =
  fixes *j1 j2* :: *tapeidx*
begin

definition *tm1* ≡ *cartesian* (*tm-bindec j1 j2*) *4*
definition *tm2* ≡ *tm1* ;; *tm-cr j1*
definition *tm3* ≡ *tm2* ;; *tm-cr j2*

lemma *tm3-eq-tm-bindecode*: *tm3* = *tm-bindecode j1 j2*
  using *tm1-def tm2-def tm3-def tm-bindecode-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *zs* :: *symbol list* **and** *k* :: *nat*
  assumes *jk*: *j1* < *k j2* < *k 0* < *j2 j1* ≠ *j2 k* = *length tps0*
  assumes *zs*: *bit-symbols zs*
  assumes *tps0*:
    tps0 ! *j1* = (⌊*zs*⌋, *1*)
    tps0 ! *j2* = (⌊[]⌋, *1*)
begin

definition *tps1* ≡ *tps0*
  [*j1* := (⌊*zs*⌋, *Suc* (*length zs*)),
   j2 := (⌊*bindecode zs*⌋, *Suc* (*length zs div 2*))]

lemma *tm1* [*transforms-intros*]:
  assumes *t* = *Suc* (*length zs*)
  shows *transforms tm1 tps0 t tps1*
  unfolding *tm1-def*
  using *transforms-cartesian-bindec assms jk tps0 zs tps1-def* **by** *blast*

definition *tps2* ≡ *tps0*
  [*j2* := (⌊*bindecode zs*⌋, *Suc* (*length zs div 2*))]

lemma *tm2* [*transforms-intros*]:
  assumes *t* = *2* ∗ *length zs* + *4*
  shows *transforms tm2 tps0 t tps2*
  unfolding *tm2-def*
proof (*tform tps*: *assms*)
  show *j1* < *length tps1*
    using *jk tps1-def* **by** *simp*
  show *clean-tape* (*tps1* ! *j1*)
    using *jk zs clean-contents-proper tps1-def* **by** *fastforce*
  show *tps2* = *tps1*[*j1* := *tps1* ! *j1* |#=| *1*]
    using *tps0 jk tps2-def tps1-def*
     by (*metis* (*no-types*, *lifting*) *fst-conv list-update-id list-update-overwrite list-update-swap nth-list-update-eq nth-list-update-neq*)
  show *t* = *Suc* (*length zs*) + (*tps1* :#: *j1* + *2*)
    using *assms*(*1*) *jk tps1-def* **by** *simp*
qed

```
definition tps3 ≡ tps0
  [j2 := (⌊bindecode zs⌋, 1)]

lemma tm3:
  assumes t = 2 * length zs + 7 + length zs div 2
  shows transforms tm3 tps0 t tps3
  unfolding tm3-def
proof (tform tps: jk tps2-def tps3-def assms)
  show clean-tape (tps2 ! j2)
    using jk bindecode-at tps2-def by simp
qed

lemma tm3':
  assumes t = 7 + 3 * length zs
  shows transforms tm3 tps0 t tps3
proof −
  have 7 + 3 * length zs ≥ 2 * length zs + 7 + length zs div 2
    by simp
  then show ?thesis
    using transforms-monotone tm3 assms tps3-def by simp
qed

end

end

lemma transforms-tm-bindecodeI [transforms-intros]:
  fixes j1 j2 :: tapeidx
  fixes tps :: tape list and zs :: symbol list and k ttt :: nat
  assumes j1 < k and j2 < k and 0 < j2 and j1 ≠ j2 and k = length tps
    and bit-symbols zs
  assumes
    tps ! j1 = (⌊zs⌋, 1)
    tps ! j2 = (⌊[]⌋, 1)
  assumes ttt = 7 + 3 * length zs
  assumes tps' = tps
    [j2 := (⌊bindecode zs⌋, 1)]
  shows transforms (tm-bindecode j1 j2) tps ttt tps'
proof −
  interpret loc: turing-machine-bindecode j1 j2 .
  show ?thesis
    using loc.tm3-eq-tm-bindecode loc.tm3' loc.tps3-def assms by simp
qed

end
```

2.11 Symbol sequence operations

```
theory Symbol-Ops
  imports Two-Four-Symbols
begin
```

While previous sections have focused on “formatted” symbol sequences for numbers and lists, in this section we devise some Turing machines dealing with “unstructured” arbitrary symbol sequences. The only “structure” that is often imposed is that of not containing a blank symbol because when reading a symbol sequence, say from the input tape, a blank would signal the end of the symbol sequence.

2.11.1 Checking for being over an alphabet

In this section we devise a Turing machine that checks if a proper symbol sequence is over a given alphabet represented by an upper bound symbol z.

abbreviation *proper-symbols-lt* :: *symbol* ⇒ *symbol list* ⇒ *bool* **where**
proper-symbols-lt z zs ≡ *proper-symbols zs* ∧ *symbols-lt z zs*

The next Turing machine checks if the symbol sequence (up until the first blank) on tape j_1 contains only symbols from $\{2, \ldots, z-1\}$, where z is a parameter of the TM, and writes to tape j_2 the number 1 or 0, representing True or False, respectively. It assumes that j_2 initially contains at most one symbol.

definition *tm-proper-symbols-lt* :: *tapeidx* ⇒ *tapeidx* ⇒ *symbol* ⇒ *machine* **where**
tm-proper-symbols-lt j1 j2 z ≡
tm-write j2 **1** ;;
WHILE [] ; *λrs. rs ! j1* ≠ □ *DO*
IF λrs. rs ! j1 < *2* ∨ *rs ! j1* ≥ *z THEN*
tm-write j2 □
ELSE
[]
ENDIF ;;
tm-right j1
DONE ;;
tm-cr j1

lemma *tm-proper-symbols-lt-tm*:
assumes *0* < *j2 j1* < *k j2* < *k* **and** *G* ≥ *4*
shows *turing-machine k G* (*tm-proper-symbols-lt j1 j2 z*)
using *assms tm-write-tm tm-right-tm tm-cr-tm Nil-tm tm-proper-symbols-lt-def*
turing-machine-loop-turing-machine turing-machine-branch-turing-machine
by *simp*

locale *turing-machine-proper-symbols-lt* =
fixes *j1 j2* :: *tapeidx* **and** *z* :: *symbol*
begin

definition *tm1* ≡ *tm-write j2* **1**
definition *tm2* ≡ *IF λrs. rs ! j1* < *2* ∨ *rs ! j1* ≥ *z THEN tm-write j2* □ *ELSE* [] *ENDIF*
definition *tm3* ≡ *tm2* ;; *tm-right j1*
definition *tm4* ≡ *WHILE* [] ; *λrs. rs ! j1* ≠ □ *DO tm3 DONE*
definition *tm5* ≡ *tm1* ;; *tm4*
definition *tm6* ≡ *tm5* ;; *tm-cr j1*

lemma *tm6-eq-tm-proper-symbols-lt*: *tm6* = *tm-proper-symbols-lt j1 j2 z*
unfolding *tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def tm-proper-symbols-lt-def*
by *simp*

context
fixes *zs* :: *symbol list* **and** *tps0* :: *tape list* **and** *k* :: *nat*
assumes *jk*: *k* = *length tps0 j1* ≠ *j2 j1* < *k j2* < *k*
and *zs*: *proper-symbols zs*
and *tps0*:
tps0 ! *j1* = (⌊*zs*⌋, *1*)
tps0 ! *j2* = (⌊[]⌋, *1*)
begin

definition *tps1 t* ≡ *tps0*
[*j1* := (⌊*zs*⌋, *Suc t*),
j2 := (*if proper-symbols-lt z* (*take t zs*) *then* ⌊[**1**]⌋ *else* ⌊[]⌋, *1*)]

lemma *tm1* [*transforms-intros*]: *transforms tm1 tps0 1* (*tps1 0*)
unfolding *tm1-def*
proof (*tform tps*: *jk tps0*)
have (*if proper-symbols-lt z* (*take 0 zs*) *then* ⌊[**1**]⌋ *else* ⌊[]⌋, *1*) = (⌊[**1**]⌋, *1*)
by *simp*
moreover have *tps0* ! *j2* |:=| **1** = (⌊[**1**]⌋, *1*)
using *tps0*(*2*) *contents-def* **by** *auto*
moreover have *tps0*[*j1* := (⌊*zs*⌋, *Suc 0*)] = *tps0*
using *tps0*(*1*) **by** (*metis One-nat-def list-update-id*)

```
  ultimately show tps1 0 = tps0[j2 := tps0 ! j2 |:=| 1]
    unfolding tps1-def by auto
qed

definition tps2 t ≡ tps0
  [j1 := (⌊zs⌋, Suc t),
   j2 := (if proper-symbols-lt z (take (Suc t) zs) then ⌊[1]⌋ else ⌊[]⌋, 1)]

lemma tm2 [transforms-intros]:
  assumes t < length zs
  shows transforms tm2 (tps1 t) 3 (tps2 t)
  unfolding tm2-def
proof (tform tps: jk tps1-def)
  have tps1 t ! j1 = (⌊zs⌋, Suc t)
    using tps1-def jk by simp
  moreover have read (tps1 t) ! j1 = tps1 t :.: j1
    using tapes-at-read' jk tps1-def by (metis (no-types, lifting) length-list-update)
  ultimately have *: read (tps1 t) ! j1 = zs ! t
    using contents-inbounds assms(1) by simp
  have j2: tps1 t ! j2 = (if proper-symbols-lt z (take t zs) then ⌊[1]⌋ else ⌊[]⌋, 1)
    using tps1-def jk by simp
  show tps2 t = (tps1 t)[j2 := tps1 t ! j2 |:=| □] if read (tps1 t) ! j1 < 2 ∨ z ≤ read (tps1 t) ! j1
  proof -
    have c3: (⌊[1]⌋, 1) |:=| □ = (⌊[]⌋, 1)
      using contents-def by auto
    have (if proper-symbols-lt z (take t zs) then ⌊[1]⌋ else ⌊[]⌋, 1) |:=| □ =
        (if proper-symbols-lt z (take (Suc t) zs) then ⌊[1]⌋ else ⌊[]⌋, 1)
    proof (cases proper-symbols-lt z (take t zs))
      case True
      have zs ! t < 2 ∨ z ≤ zs ! t
        using that * by simp
      then have ¬ proper-symbols-lt z (take (Suc t) zs)
        using assms(1) by auto
      then show ?thesis
        using c3 by auto
    next
      case False
      then have ¬ proper-symbols-lt z (take (Suc t) zs)
        by auto
      then show ?thesis
        using c3 False by auto
    qed
    then have tps1 t ! j2 |:=| □ = (if proper-symbols-lt z (take (Suc t) zs) then ⌊[1]⌋ else ⌊[]⌋, 1)
      using j2 by simp
    then show tps2 t = (tps1 t)[j2 := tps1 t ! j2 |:=| □]
      unfolding tps2-def tps1-def using c3 jk(1,4) by simp
  qed
  show tps2 t = tps1 t if ¬ (read (tps1 t) ! j1 < 2 ∨ z ≤ read (tps1 t) ! j1)
  proof -
    have 1: zs ! t ≥ 2 ∧ z > zs ! t
      using that * by simp
    show tps2 t = tps1 t
    proof (cases proper-symbols-lt z (take t zs))
      case True
      have proper-symbols-lt z (take (Suc t) zs)
        using True 1 assms(1) zs by (metis length-take less-antisym min-less-iff-conj nth-take)
      then show ?thesis
        using tps1-def tps2-def jk by simp
    next
      case False
      then have ¬ proper-symbols-lt z (take (Suc t) zs)
        by auto
      then show ?thesis
```

using *tps1-def tps2-def jk False* **by** *auto*
qed
qed
qed

lemma *tm3* [*transforms-intros*]:
assumes $t < \textit{length zs}$
shows *transforms tm3* (*tps1 t*) *4* (*tps1* (*Suc t*))
unfolding *tm3-def*
proof (*tform tps*: *assms jk tps2-def*)
have *tps2 t* ! *j1* |+| *1* = (⌊*zs*⌋, *Suc* (*Suc t*))
using *tps2-def jk* **by** *simp*
then show *tps1* (*Suc t*) = (*tps2 t*)[*j1* := *tps2 t* ! *j1* |+| *1*]
unfolding *tps1-def tps2-def*
by (*metis* (*no-types*, *lifting*) *jk*(*2*) *list-update-overwrite list-update-swap*)
qed

lemma *tm4* [*transforms-intros*]:
assumes *ttt* = *1* + *6* ∗ *length zs*
shows *transforms tm4* (*tps1 0*) *ttt* (*tps1* (*length zs*))
unfolding *tm4-def*
proof (*tform time*: *assms*)
show *read* (*tps1 t*) ! *j1* ≠ □ **if** $t < \textit{length zs}$ **for** *t*
proof −
have *tps1 t* ! *j1* = (⌊*zs*⌋, *Suc t*)
using *tps1-def jk* **by** *simp*
moreover have *read* (*tps1 t*) ! *j1* = *tps1 t* :.: *j1*
using *tapes-at-read′ jk tps1-def* **by** (*metis* (*no-types*, *lifting*) *length-list-update*)
ultimately have *read* (*tps1 t*) ! *j1* = *zs* ! *t*
using *contents-inbounds that* **by** *simp*
then show *?thesis*
using *zs that* **by** *auto*
qed
show ¬ *read* (*tps1* (*length zs*)) ! *j1* ≠ □
proof −
have *tps1* (*length zs*) ! *j1* = (⌊*zs*⌋, *Suc* (*length zs*))
using *tps1-def jk* **by** *simp*
moreover have *read* (*tps1* (*length zs*)) ! *j1* = *tps1* (*length zs*) :.: *j1*
using *tapes-at-read′ jk tps1-def* **by** (*metis* (*no-types*, *lifting*) *length-list-update*)
ultimately show *?thesis*
by *simp*
qed
qed

lemma *tm5* [*transforms-intros*]:
assumes *ttt* = *2* + *6* ∗ *length zs*
shows *transforms tm5 tps0 ttt* (*tps1* (*length zs*))
unfolding *tm5-def*
by (*tform time*: *assms*)

definition *tps5* ≡ *tps0*
[*j1* := (⌊*zs*⌋, *1*),
j2 := (*if proper-symbols-lt z zs then* ⌊[**1**]⌋ *else* ⌊[]⌋, *1*)]

definition *tps5′* ≡ *tps0*
[*j2* := (*if proper-symbols-lt z zs then* ⌊[**1**]⌋ *else* ⌊[]⌋, *1*)]

lemma *tm6*:
assumes *ttt* = *5* + *7* ∗ *length zs*
shows *transforms tm6 tps0 ttt tps5′*
unfolding *tm6-def*
proof (*tform time*: *assms tps1-def jk*)
have ∗: *tps1* (*length zs*) ! *j1* = (⌊*zs*⌋, *Suc* (*length zs*))

using *tps1-def jk* **by** *simp*
show *clean-tape* (*tps1* (*length zs*) ! *j1*)
using ∗ *zs* **by** *simp*
have *tps5* = (*tps1* (*length zs*))[*j1* := (⌊*zs*⌋, *Suc* (*length zs*)) |#=| *1*]
unfolding *tps5-def tps1-def* **by** (*simp add*: *list-update-swap*[*OF jk*(*2*)])
then have *tps5* = (*tps1* (*length zs*))[*j1* := *tps1* (*length zs*) ! *j1* |#=| *1*]
using ∗ **by** *simp*
moreover have *tps5′* = *tps5*
using *tps5′-def tps5-def tps0 jk* **by** (*metis list-update-id*)
ultimately show *tps5′* = (*tps1* (*length zs*))[*j1* := *tps1* (*length zs*) ! *j1* |#=| *1*]
by *simp*
qed

definition *tps6* ≡ *tps0*
[*j2* := (⌊*proper-symbols-lt z zs*⌋$_B$, *1*)]

lemma *tm6′*:
assumes *ttt* = *5* + *7* ∗ *length zs*
shows *transforms tm6 tps0 ttt tps6*
proof −
have *tps6* = *tps5′*
using *tps6-def tps5′-def canrepr-0 canrepr-1* **by** *auto*
then show *?thesis*
using *tm6 assms* **by** *simp*
qed

end

end

lemma *transforms-tm-proper-symbols-ltI* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx* **and** *z* :: *symbol*
fixes *zs* :: *symbol list* **and** *tps tps′* :: *tape list* **and** *k* :: *nat*
assumes *k* = *length tps j1* ≠ *j2 j1* < *k j2* < *k*
and *proper-symbols zs*
assumes
tps ! *j1* = (⌊*zs*⌋, *1*)
tps ! *j2* = (⌊[]⌋, *1*)
assumes *ttt* = *5* + *7* ∗ *length zs*
assumes *tps′* = *tps*
[*j2* := (⌊*proper-symbols-lt z zs*⌋$_B$, *1*)]
shows *transforms* (*tm-proper-symbols-lt j1 j2 z*) *tps ttt tps′*
proof −
interpret *loc*: *turing-machine-proper-symbols-lt j1 j2* **.**
show *?thesis*
using *assms loc.tm6-eq-tm-proper-symbols-lt loc.tps6-def loc.tm6′* **by** *simp*
qed

2.11.2 The length of the input

The Turing machine in this section reads the input tape until the first blank and increments a counter on tape *j* for every symbol read. In the end it performs a carriage return on the input tape, and tape *j* contains the length of the input as binary number. For this to work, tape *j* must initially be empty.

lemma *proper-tape-read*:
assumes *proper-symbols zs*
shows |.| (⌊*zs*⌋, *i*) = □ ⟷ *i* > *length zs*
proof −
have |.| (⌊*zs*⌋, *i*) = □ **if** *i* > *length zs* **for** *i*
using *that contents-outofbounds* **by** *simp*
moreover have |.| (⌊*zs*⌋, *i*) ≠ □ **if** *i* ≤ *length zs* **for** *i*
using *that contents-inbounds assms contents-def proper-symbols-ne0* **by** *simp*
ultimately show *?thesis*
by (*meson le-less-linear*)

qed

definition *tm-length-input* :: *tapeidx* ⇒ *machine* **where**
  tm-length-input j ≡
    WHILE [] ; *λrs. rs* ! *0* ≠ □ *DO*
      tm-incr j ;;
      tm-right 0
    DONE ;;
    tm-cr 0

lemma *tm-length-input-tm*:
  assumes *G* ≥ *4* **and** *0* < *j* **and** *j* < *k*
  shows *turing-machine k G* (*tm-length-input j*)
  using *tm-length-input-def tm-incr-tm assms Nil-tm tm-right-tm tm-cr-tm*
  by (*simp add*: *turing-machine-loop-turing-machine*)

locale *turing-machine-length-input* =
  fixes *j* :: *tapeidx*
begin

definition *tmL1* ≡ *tm-incr j*
definition *tmL2* ≡ *tmL1* ;; *tm-right 0*
definition *tm1* ≡ *WHILE* [] ; *λrs. rs* ! *0* ≠ □ *DO tmL2 DONE*
definition *tm2* ≡ *tm1* ;; *tm-cr 0*

lemma *tm2-eq-tm-length-input*: *tm2* = *tm-length-input j*
  unfolding *tm2-def tm1-def tmL2-def tmL1-def tm-length-input-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *zs* :: *symbol list*
  assumes *jk*: *0* < *j j* < *k length tps0* = *k*
    and *zs*: *proper-symbols zs*
    and *tps0*:
      tps0 ! *0* = (⌊*zs*⌋, *1*)
      tps0 ! *j* = (⌊*0*⌋$_N$, *1*)
begin

definition *tpsL* :: *nat* ⇒ *tape list* **where**
  tpsL t ≡ *tps0*[*0* := (⌊*zs*⌋, *1* + *t*), *j* := (⌊*t*⌋$_N$, *1*)]

lemma *tpsL-eq-tps0*: *tpsL 0* = *tps0*
  using *tpsL-def tps0 jk* **by** (*metis One-nat-def list-update-id plus-1-eq-Suc*)

definition *tpsL1* :: *nat* ⇒ *tape list* **where**
  tpsL1 t ≡ *tps0*[*0* := (⌊*zs*⌋, *1* + *t*), *j* := (⌊*Suc t*⌋$_N$, *1*)]

definition *tpsL2* :: *nat* ⇒ *tape list* **where**
  tpsL2 t ≡ *tps0*[*0* := (⌊*zs*⌋, *1* + *Suc t*), *j* := (⌊*Suc t*⌋$_N$, *1*)]

lemma *tmL1* [*transforms-intros*]:
  assumes *t* < *length zs* **and** *ttt* = *5* + *2* ∗ *nlength t*
  shows *transforms tmL1* (*tpsL t*) *ttt* (*tpsL1 t*)
  unfolding *tmL1-def*
  by (*tform tps*: *assms*(*1*) *tpsL-def tpsL1-def tps0 jk time*: *assms*(*2*))

lemma *tmL2*:
  assumes *t* < *length zs* **and** *ttt* = *6* + *2* ∗ *nlength t*
  shows *transforms tmL2* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
  unfolding *tmL2-def*
proof (*tform tps*: *assms*(*1*) *tpsL-def tpsL1-def tps0 jk time*: *assms*(*2*))
  have *tpsL1 t* ! *0* = (⌊*zs*⌋, *1* + *t*)
    using *tpsL2-def tpsL1-def jk tps0* **by** *simp*
  then have *tpsL2 t* = (*tpsL1 t*)[*0* := *tpsL1 t* ! *0* |#=| *Suc* (*tpsL1 t* :#: *0*)]

using *tpsL2-def tpsL1-def jk tps0*
by (*smt* (*verit*) *fstI list-update-overwrite list-update-swap nat-neq-iff plus-1-eq-Suc prod.sel*(*2*))
then show *tpsL* (*Suc t*) = (*tpsL1 t*)[*0* := *tpsL1 t* ! *0* |+| *1*]
using *tpsL2-def tpsL-def tpsL1-def jk tps0* **by** *simp*
qed

lemma *tmL2′*:
assumes *t* < *length zs* **and** *ttt* = *6* + *2* ∗ *nlength* (*length zs*)
shows *transforms tmL2* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
proof −
have *6* + *2* ∗ *nlength t* ≤ *6* + *2* ∗ *nlength* (*length zs*)
using *assms*(*1*) *nlength-mono* **by** *simp*
then show *?thesis*
using *assms tmL2 transforms-monotone* **by** *blast*
qed

lemma *tm1*:
assumes *ttt* = *length zs* ∗ (*8* + *2* ∗ *nlength* (*length zs*)) + *1*
shows *transforms tm1* (*tpsL 0*) *ttt* (*tpsL* (*length zs*))
unfolding *tm1-def*
proof (*tform*)
let *?t* = *6* + *2* ∗ *nlength* (*length zs*)
show ⋀*t*. *t* < *length zs* ⟹ *transforms tmL2* (*tpsL t*) *?t* (*tpsL* (*Suc t*))
using *tmL2′* **by** *simp*
have ∗: *tpsL t* ! *0* = (⌊*zs*⌋, *Suc t*) **for** *t*
using *tpsL-def jk* **by** *simp*
then show ⋀*t*. *t* < *length zs* ⟹ *read* (*tpsL t*) ! *0* ≠ □
using *proper-tape-read*[*OF zs*] *tpsL-def jk tapes-at-read′*
by (*metis length-list-update less-Suc-eq-0-disj not-less-eq*)
show ¬ *read* (*tpsL* (*length zs*)) ! *0* ≠ □
using *proper-tape-read*[*OF zs*] *tpsL-def jk tapes-at-read′* ∗
by (*metis length-list-update less-Suc-eq-0-disj less-imp-Suc-add nat-neq-iff not-less-less-Suc-eq*)
show *length zs* ∗ (*6* + *2* ∗ *nlength* (*length zs*) + *2*) + *1* ≤ *ttt*
using *assms* **by** *simp*
qed

lemma *tmL′* [*transforms-intros*]:
assumes *ttt* = *10* ∗ *length zs* ^ *2* + *1*
shows *transforms tm1* (*tpsL 0*) *ttt* (*tpsL* (*length zs*))
proof −
let *?ttt* = *length zs* ∗ (*8* + *2* ∗ *nlength* (*length zs*)) + *1*
have *?ttt* ≤ *length zs* ∗ (*8* + *2* ∗ *length zs*) + *1*
using *nlength-le-n* **by** *simp*
also have ... ≤ *8* ∗ *length zs* + *2* ∗ *length zs* ^ *2* + *1*
by (*simp add*: *add-mult-distrib2 power2-eq-square*)
also have ... ≤ *10* ∗ *length zs* ^ *2* + *1*
using *linear-le-pow* **by** *simp*
finally have *?ttt* ≤ *10* ∗ *length zs* ^ *2* + *1* **.**
then show *?thesis*
using *tm1 assms transforms-monotone* **by** *simp*
qed

definition *tps2* :: *tape list* **where**
tps2 ≡ *tps0*[*0* := (⌊*zs*⌋, *1*), *j* := (⌊*length zs*⌋$_N$, *1*)]

lemma *tm2*:
assumes *ttt* = *10* ∗ *length zs* ^ *2* + *length zs* + *4*
shows *transforms tm2* (*tpsL 0*) *ttt tps2*
unfolding *tm2-def*
proof (*tform time*: *assms tpsL-def jk tps*: *tpsL-def tpsL1-def tps0 jk*)
show *clean-tape* (*tpsL* (*length zs*) ! *0*)
using *tpsL-def jk clean-contents-proper*[*OF zs*] **by** *simp*
have *tpsL* (*length zs*) ! *0* = (⌊*zs*⌋, *Suc* (*length zs*))

using *tpsL-def jk* **by** *simp*
then show *tps2* = (*tpsL* (*length zs*))[*0* := *tpsL* (*length zs*) ! *0* |#=| *1*]
using *tps2-def tpsL-def jk* **by** (*simp add*: *list-update-swap-less*)
qed

definition *tps2′* :: *tape list* **where**
tps2′ ≡ *tps0*[*j* := (⌊*length zs*⌋$_N$, *1*)]

lemma *tm2′*:
assumes *ttt* = *11* ∗ *length zs* ^ *2* + *4*
shows *transforms tm2 tps0 ttt tps2′*
proof −
have *10* ∗ *length zs* ^ *2* + *length zs* + *4* ≤ *ttt*
using *assms linear-le-pow*[*of 2 length zs*] **by** *simp*
moreover have *tps2* = *tps2′*
using *tps2-def tps2′-def jk tps0* **by** (*metis list-update-id*)
ultimately show *?thesis*
using *transforms-monotone tm2 tpsL-eq-tps0* **by** *simp*
qed

end

end

lemma *transforms-tm-length-inputI* [*transforms-intros*]:
fixes *j* :: *tapeidx*
fixes *tps tps′* :: *tape list* **and** *k* :: *nat* **and** *zs* :: *symbol list*
assumes *0* < *j j* < *k length tps* = *k*
and *proper-symbols zs*
assumes
tps ! *0* = (⌊*zs*⌋, *1*)
tps ! *j* = (⌊*0*⌋$_N$, *1*)
assumes *ttt* = *11* ∗ *length zs* ^ *2* + *4*
assumes *tps′* = *tps*
[*j* := (⌊*length zs*⌋$_N$, *1*)]
shows *transforms* (*tm-length-input j*) *tps ttt tps′*
proof −
interpret *loc*: *turing-machine-length-input j* **.**
show *?thesis*
using *loc.tm2′ loc.tps2′-def loc.tm2-eq-tm-length-input assms* **by** *simp*
qed

2.11.3 Whether the length is even

The next Turing machines reads the symbols on tape j_1 until the first blank and alternates between numbers 0 and 1 on tape j_2. Then tape j_2 contains the parity of the length of the symbol sequence on tape j_1. Finally, the TM flips the output once more, so that tape j_2 contains a Boolean indicating whether the length was even or not. We assume tape j_2 is initially empty, that is, represents the number 0.

definition *tm-even-length* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-even-length j1 j2 ≡
WHILE [] ; λ*rs. rs* ! *j1* ≠ □ *DO*
tm-not j2 ;;
tm-right j1
DONE ;;
tm-not j2 ;;
tm-cr j1

lemma *tm-even-length-tm*:
assumes *k* ≥ *2* **and** *G* ≥ *4* **and** *j1* < *k 0* < *j2 j2* < *k*
shows *turing-machine k G* (*tm-even-length j1 j2*)
using *tm-even-length-def tm-right-tm tm-not-tm Nil-tm assms tm-cr-tm turing-machine-loop-turing-machine*
by *simp*

locale *turing-machine-even-length* =
  fixes *j1 j2* :: *tapeidx*
begin

definition *tmB* ≡ *tm-not j2* ;; *tm-right j1*
definition *tmL* ≡ *WHILE* [] ; *λrs. rs* ! *j1* ≠ □ *DO tmB DONE*
definition *tm2* ≡ *tmL* ;; *tm-not j2*
definition *tm3* ≡ *tm2* ;; *tm-cr j1*

lemma *tm3-eq-tm-even-length*: *tm3* = *tm-even-length j1 j2*
  unfolding *tm3-def tm2-def tmL-def tmB-def tm-even-length-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *zs* :: *symbol list*
  assumes *zs*: *proper-symbols zs*
  assumes *jk*: *j1* < *k j2* < *k j1* ≠ *j2 length tps0* = *k*
  assumes *tps0*:
    tps0 ! *j1* = (⌊*zs*⌋, *1*)
    tps0 ! *j2* = (⌊*0*⌋$_N$, *1*)
begin

definition *tpsL* :: *nat* ⇒ *tape list* **where**
  tpsL t ≡ *tps0*
    [*j1* := (⌊*zs*⌋, *Suc t*),
     j2 := (⌊*odd t*⌋$_B$, *1*)]

lemma *tpsL0*: *tpsL 0* = *tps0*
  unfolding *tpsL-def* **using** *tps0 jk* **by** (*metis* (*mono-tags*, *opaque-lifting*) *One-nat-def even-zero list-update-id*)

lemma *tmL2* [*transforms-intros*]: *transforms tmB* (*tpsL t*) *4* (*tpsL* (*Suc t*))
  unfolding *tmB-def*
proof (*tform tps*: *tpsL-def jk*)
  have (*tpsL t*)
    [*j2* := (⌊(*if odd t then 1 else 0* :: *nat*) ≠ *1*⌋$_B$, *1*),
     j1 := (*tpsL t*)[*j2* := (⌊ (*if odd t then 1 else 0* :: *nat*) ≠ *1* ⌋$_B$, *1*)] ! *j1* |+| *1*] =
    (*tpsL t*)
    [*j2* := (⌊*odd* (*Suc t*)⌋$_B$, *1*),
     j1 := (*tpsL t*) ! *j1* |+| *1*]
    using *jk* **by** *simp*
  also have ... = (*tpsL t*)
    [*j2* := (⌊*odd* (*Suc t*)⌋$_B$, *1*),
     j1 := (⌊*zs*⌋, *Suc* (*Suc t*))]
    using *tpsL-def jk* **by** *simp*
  also have ... = (*tpsL t*)
    [*j1* := (⌊*zs*⌋, *Suc* (*Suc t*)),
     j2 := (⌊*odd* (*Suc t*)⌋$_B$, *1*)]
    using *jk* **by** (*simp add*: *list-update-swap*)
  also have ... = *tps0*
    [*j1* := (⌊*zs*⌋, *Suc* (*Suc t*)),
     j2 := (⌊*odd* (*Suc t*)⌋$_B$, *1*)]
    using *jk tpsL-def* **by** (*simp add*: *list-update-swap*)
  also have ... = *tpsL* (*Suc t*)
    using *tpsL-def* **by** *simp*
  finally show *tpsL* (*Suc t*) = (*tpsL t*)
    [*j2* := (⌊(*if odd t then 1 else 0* :: *nat*) ≠ *1*⌋$_B$, *1*),
     j1 := (*tpsL t*)[*j2* := (⌊(*if odd t then 1 else 0* :: *nat*) ≠ *1*⌋$_B$, *1*)] ! *j1* |+| *1*]
    by *simp*
qed

lemma *tmL*:
  assumes *ttt* = *6* ∗ *length zs* + *1*
  shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsL* (*length zs*))
  unfolding *tmL-def*

proof (*tform time*: *assms*)
  have *read* (*tpsL t*) ! *j1* = *tpsL t* :.: *j1* **for** *t*
    using *tpsL-def tapes-at-read′ jk*
    by (*metis* (*no-types*, *lifting*) *length-list-update*)
  then have *read* (*tpsL t*) ! *j1* = ⌊*zs*⌋ (*Suc t*) **for** *t*
    using *tpsL-def jk* **by** *simp*
  then show $\bigwedge t.\ t < length\ zs \Longrightarrow read\ (tpsL\ t)\ !\ j1 \neq \Box$ **and** $\neg\ read\ (tpsL\ (length\ zs))\ !\ j1 \neq \Box$
    using *zs* **by** *auto*
qed

lemma *tmL′* [*transforms-intros*]:
  assumes *ttt* = *6* ∗ *length zs* + *1*
  shows *transforms tmL tps0 ttt* (*tpsL* (*length zs*))
  using *assms tmL tpsL0* **by** *simp*

definition *tps2* :: *tape list* **where**
  tps2 ≡ *tps0*
    [*j1* := (⌊*zs*⌋, *Suc* (*length zs*)),
     j2 := (⌊*even* (*length zs*) ⌋$_B$, *1*)]

lemma *tm2* [*transforms-intros*]:
  assumes *ttt* = *6* ∗ *length zs* + *4*
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def*
proof (*tform tps*: *tpsL-def jk time*: *assms*)
  show *tps2* = (*tpsL* (*length zs*))[*j2* := (⌊(*if odd* (*length zs*) *then 1 else 0* :: *nat*) ≠ *1*⌋$_B$, *1*)]
    unfolding *tps2-def tpsL-def* **by** (*simp add*: *list-update-swap*)
qed

definition *tps3* :: *tape list* **where**
  tps3 ≡ *tps0*
    [*j1* := (⌊*zs*⌋, *1*),
     j2 := (⌊*even* (*length zs*)⌋$_B$, *1*)]

lemma *tm3*:
  assumes *ttt* = *7* ∗ *length zs* + *7*
  shows *transforms tm3 tps0 ttt tps3*
  unfolding *tm3-def*
proof (*tform tps*: *tps2-def jk time*: *assms*)
  show *clean-tape* (*tps2* ! *j1*)
    using *tps2-def jk zs clean-contents-proper* **by** *simp*
  have *tps2* ! *j1* |#=| *1* = (⌊*zs*⌋, *1*)
    using *tps2-def jk* **by** *simp*
  then show *tps3* = *tps2*[*j1* := *tps2* ! *j1* |#=| *1*]
    unfolding *tps3-def tps2-def* **using** *jk* **by** (*simp add*: *list-update-swap*)
  show *ttt* = *6* ∗ *length zs* + *4* + (*tps2* :#: *j1* + *2*)
    using *assms tps2-def jk* **by** *simp*
qed

definition *tps3′* :: *tape list* **where**
  tps3′ ≡ *tps0*
    [*j2* := (⌊*even* (*length zs*)⌋$_B$, *1*)]

lemma *tps3′*: *tps3′* = *tps3*
  using *tps3′-def tps3-def tps0* **by** (*metis list-update-id*)

lemma *tm3′*:
  assumes *ttt* = *7* ∗ *length zs* + *7*
  shows *transforms tm3 tps0 ttt tps3′*
  using *tps3′ tm3 assms* **by** *simp*

end

end

lemma *transforms-tm-even-lengthI* [*transforms-intros*]:
  fixes *j1 j2* :: *tapeidx*
  fixes *tps tps'* :: *tape list* **and** *k* :: *nat* **and** *zs* :: *symbol list*
  assumes $j1 < k$ $j2 < k$ $j1 \neq j2$
    and *proper-symbols zs*
    and *length tps* = *k*
  assumes
    tps ! *j1* = (⌊*zs*⌋, *1*)
    tps ! *j2* = (⌊*0*⌋$_N$, *1*)
  assumes *tps'* = *tps*
    [*j2* := (⌊*even* (*length zs*)⌋$_B$, *1*)]
  assumes *ttt* = *7* ∗ *length zs* + *7*
  shows *transforms* (*tm-even-length j1 j2*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-even-length j1 j2* **.**
  show *?thesis*
    using *assms loc.tps3'-def loc.tm3' loc.tm3-eq-tm-even-length* **by** *simp*
qed

2.11.4 Checking for ends-with or empty

The next Turing machine implements a slightly idiosyncratic operation that we use in the next section for checking if a symbol sequence represents a list of numbers. The operation consists in checking if the symbol sequence on tape j_1 either is empty or ends with the symbol z, which is another parameter of the TM. If the condition is met, the number 1 is written to tape j_2, otherwise the number 0.

definition *tm-empty-or-endswith* :: *tapeidx* ⇒ *tapeidx* ⇒ *symbol* ⇒ *machine* **where**
  tm-empty-or-endswith j1 j2 z ≡
    tm-right-until j1 {□} ;;
    tm-left j1 ;;
    IF λ*rs*. *rs* ! *j1* ∈ {▷, *z*} *THEN*
      tm-setn j2 1
    ELSE
      []
    ENDIF ;;
    tm-cr j1

lemma *tm-empty-or-endswith-tm*:
  assumes $k \geq 2$ **and** $G \geq 4$ **and** $0 < j2$ **and** $j1 < k$ **and** $j2 < k$
  shows *turing-machine k G* (*tm-empty-or-endswith j1 j2 z*)
  using *assms Nil-tm tm-right-until-tm tm-left-tm tm-setn-tm tm-cr-tm*
    turing-machine-branch-turing-machine tm-empty-or-endswith-def
  by *simp*

locale *turing-machine-empty-or-endswith* =
  fixes *j1 j2* :: *tapeidx* **and** *z* :: *symbol*
begin

definition *tm1* ≡ *tm-right-until j1* {□}
definition *tm2* ≡ *tm1* ;; *tm-left j1*
definition *tmI* ≡ *IF* λ*rs*. *rs* ! *j1* ∈ {▷, *z*} *THEN tm-setn j2 1 ELSE* [] *ENDIF*
definition *tm3* ≡ *tm2* ;; *tmI*
definition *tm4* ≡ *tm3* ;; *tm-cr j1*

lemma *tm4-eq-tm-empty-or-endswith*: *tm4* = *tm-empty-or-endswith j1 j2 z*
  unfolding *tm4-def tm3-def tmI-def tm2-def tm1-def tm-empty-or-endswith-def*
  by *simp*

context
  fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *zs* :: *symbol list*
  assumes *jk*: $j1 \neq j2$ $j1 < k$ $j2 < k$ *length tps0* = *k*
    and *zs*: *proper-symbols zs*

and *tps0*:
tps0 ! *j1* = (⌊*zs*⌋, *1*)
tps0 ! *j2* = (⌊*0*⌋$_N$, *1*)
begin

definition *tps1* :: *tape list* **where**
tps1 ≡ *tps0*
[*j1* := (⌊*zs*⌋, *Suc* (*length zs*))]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *Suc* (*length zs*)
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
proof (*tform time*: *assms tps*: *tps0 tps1-def jk*)
show *rneigh* (*tps0* ! *j1*) {*0*} (*length zs*)
proof (*rule rneighI*)
show (*tps0* ::: *j1*) (*tps0* :#: *j1* + *length zs*) ∈ {*0*}
using *tps0* **by** *simp*
show ⋀*n'*. *n'* < *length zs* ⟹ (*tps0* ::: *j1*) (*tps0* :#: *j1* + *n'*) ∉ {*0*}
using *zs tps0* **by** *auto*
qed
qed

definition *tps2* :: *tape list* **where**
tps2 ≡ *tps0*
[*j1* := (⌊*zs*⌋, *length zs*)]

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *2* + *length zs*
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def*
by (*tform time*: *assms tps*: *tps1-def tps2-def jk*)

definition *tps3* :: *tape list* **where**
tps3 ≡ *tps0*
[*j1* := (⌊*zs*⌋, *length zs*),
j2 := (⌊*zs* = [] ∨ *last zs* = *z*⌋$_B$, *1*)]

lemma *tmI* [*transforms-intros*]: *transforms tmI tps2 14 tps3*
unfolding *tmI-def*
proof (*tform tps*: *tps0 tps2-def jk*)
have ∗: *read tps2* ! *j1* = ⌊*zs*⌋ (*length zs*)
using *tps2-def jk tapes-at-read'*[*of j1 tps2*] **by** *simp*

show *tps3* = *tps2*[*j2* := (⌊*1*⌋$_N$, *1*)] **if** *read tps2* ! *j1* ∈ {▷, *z*}
proof −
have *zs* = [] ∨ *last zs* = *z*
using *that* ∗ *contents-inbounds zs*
by (*metis diff-less dual-order.refl insert-iff last-conv-nth length-greater-0-conv proper-symbols-ne1 singletonD zero-less-one*)
then have (*if zs* = [] ∨ *last zs* = *z then 1 else 0*) = *1*
by *simp*
then show *?thesis*
using *tps2-def tps3-def jk* **by** (*smt* (*verit*, *best*))
qed
show *tps3* = *tps2* **if** *read tps2* ! *j1* ∉ {▷, *z*}
proof −
have ¬ (*zs* = [] ∨ *last zs* = *z*)
using *that* ∗ *contents-inbounds zs*
by (*metis contents-at-0 dual-order.refl insertCI last-conv-nth length-greater-0-conv list.size*(*3*))
then have (*if zs* = [] ∨ *last zs* = *z then 1 else 0*) = *0*
by *simp*
then show *?thesis*

using *tps2-def tps3-def jk tps0* **by** (*smt* (*verit*, *best*) *list-update-id nth-list-update-neq*)
qed
show $10 + 2 * nlength\ 0 + 2 * nlength\ 1 + 2 \leq 14$
using *nlength-1-simp* **by** *simp*
qed

lemma *tm3* [*transforms-intros*]:
assumes $ttt = 16 + length\ zs$
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def* **by** (*tform tps*: *assms*)

definition *tps4* :: *tape list* **where**
$tps4 \equiv tps0$
$[j2 := (\lfloor zs = [] \vee last\ zs = z \rfloor_B,\ 1)]$

lemma *tm4*:
assumes $ttt = 18 + 2 * length\ zs$
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform time*: *assms tps3-def jk tps*: *tps3-def jk zs*)
have $tps3\ !\ j1\ |\#=|\ 1 = (\lfloor zs \rfloor,\ 1)$
using *tps3-def jk zs* **by** *simp*
then show $tps4 = tps3[j1 := tps3\ !\ j1\ |\#=|\ 1]$
using *tps4-def tps3-def jk tps0*(*1*) **by** (*metis list-update-id list-update-overwrite list-update-swap*)
qed

end

end

lemma *transforms-tm-empty-or-endswithI* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx* **and** *z* :: *symbol*
fixes *tps tps'* :: *tape list* **and** *k* :: *nat* **and** *zs* :: *symbol list*
assumes $j1 \neq j2\ j1 < k\ j2 < k$
and $length\ tps = k$
and *proper-symbols zs*
assumes
$tps\ !\ j1 = (\lfloor zs \rfloor,\ 1)$
$tps\ !\ j2 = (\lfloor 0 \rfloor_N,\ 1)$
assumes $ttt = 18 + 2 * length\ zs$
assumes $tps' = tps$
$[j2 := (\lfloor zs = [] \vee last\ zs = z \rfloor_B,\ 1)]$
shows *transforms* (*tm-empty-or-endswith j1 j2 z*) *tps ttt tps'*
proof −
interpret *loc*: *turing-machine-empty-or-endswith j1 j2 z* **.**
show *?thesis*
using *assms loc.tps4-def loc.tm4 loc.tm4-eq-tm-empty-or-endswith* **by** *simp*
qed

2.11.5 Stripping trailing symbols

Stripping the symbol z from the end of a symbol sequence zs means:

definition *rstrip* :: *symbol* ⇒ *symbol list* ⇒ *symbol list* **where**
$rstrip\ z\ zs \equiv take\ (LEAST\ i.\ i \leq length\ zs \wedge set\ (drop\ i\ zs) \subseteq \{z\})\ zs$

lemma *length-rstrip*: $length\ (rstrip\ z\ zs) = (LEAST\ i.\ i \leq length\ zs \wedge set\ (drop\ i\ zs) \subseteq \{z\})$
using *rstrip-def wellorder-Least-lemma*[**where** $?P=\lambda i.\ i \leq length\ zs \wedge set\ (drop\ i\ zs) \subseteq \{z\}$] **by** *simp*

lemma *length-rstrip-le*: $length\ (rstrip\ z\ zs) \leq length\ zs$
using *rstrip-def* **by** *simp*

lemma *rstrip-stripped*:
assumes $i \geq length\ (rstrip\ z\ zs)$ **and** $i < length\ zs$

```
  shows zs ! i = z
proof -
  let ?P = λi. i ≤ length zs ∧ set (drop i zs) ⊆ {z}
  have ?P (length zs)
    by simp
  then have ?P i
    using assms length-rstrip LeastI[where ?P=?P] Least-le[where ?P=?P]
    by (metis (mono-tags, lifting) dual-order.trans order-less-imp-le set-drop-subset-set-drop)
  then have set (drop i zs) ⊆ {z}
    by simp
  then show ?thesis
     using assms(2) by (metis Cons-nth-drop-Suc drop-eq-Nil2 leD list.set(2) set-empty singleton-insert-inj-eq
subset-singletonD)
qed

lemma rstrip-replicate: rstrip z (replicate n z) = []
  using rstrip-def
  by (metis (no-types, lifting) Least-eq-0 empty-replicate set-drop-subset set-replicate take-eq-Nil zero-le)

lemma rstrip-not-ex:
  assumes ¬ (∃ i<length zs. zs ! i ≠ z)
  shows rstrip z zs = []
  using assms rstrip-def by (metis in-set-conv-nth replicate-eqI rstrip-replicate)

lemma
  assumes ∃ i<length zs. zs ! i ≠ z
  shows rstrip-ex-length: length (rstrip z zs) > 0
    and rstrip-ex-last: last (rstrip z zs) ≠ z
proof -
  let ?P = λi. i ≤ length zs ∧ set (drop i zs) ⊆ {z}
  obtain i where i: i < length zs zs ! i ≠ z
    using assms by auto
  then have ¬ set (drop i zs) ⊆ {z}
   by (metis Cons-nth-drop-Suc drop-eq-Nil2 leD list.set(2) set-empty singleton-insert-inj-eq' subset-singletonD)
  then have ¬ set (drop 0 zs) ⊆ {z}
   by (metis drop.simps(1) drop-0 set-drop-subset set-empty subset-singletonD)
  then show len: length (rstrip z zs) > 0
    using length-rstrip by (metis (no-types, lifting) LeastI bot.extremum drop-all dual-order.refl gr0I list.set(1))
  let ?j = length (rstrip z zs) - 1
  have 3: ?j < length (rstrip z zs)
    using len by simp
  then have 4: ?j < Least ?P
    using length-rstrip by simp
  have 5: ?P (length (rstrip z zs))
    using LeastI-ex[of ?P] length-rstrip by fastforce
  show last (rstrip z zs) ≠ z
  proof (rule ccontr)
    assume ¬ last (rstrip z zs) ≠ z
    then have last (rstrip z zs) = z
      by simp
    then have rstrip z zs ! ?j = z
      using len by (simp add: last-conv-nth)
    then have 2: zs ! ?j = z
      using len length-rstrip rstrip-def by auto
    have ?P ?j
    proof -
      have ?j ≤ length zs
        using 3 length-rstrip-le by (meson le-eq-less-or-eq order-less-le-trans)
      moreover have set (drop ?j zs) ⊆ {z}
        using 5 3 2
           by (metis Cons-nth-drop-Suc One-nat-def Suc-pred insert-subset len list.simps(15) order-less-le-trans
set-eq-subset)
      ultimately show ?thesis
```

by *simp*
qed
then show *False*
using *4 Least-le*[*of ?P*] **by** *fastforce*
qed
qed

A Turing machine stripping the non-blank, non-start symbol z from a proper symbol sequence works in the obvious way. First it moves to the end of the symbol sequence, that is, to the first blank. Then it moves left to the first non-z symbol thereby overwriting every symbol with a blank. Finally it performs a "carriage return".

definition *tm-rstrip* :: *symbol* ⇒ *tapeidx* ⇒ *machine* **where**
tm-rstrip z j ≡
tm-right-until j {□} ;;
tm-left j ;;
tm-lconst-until j j (*UNIV* − {*z*}) □ ;;
tm-cr j

lemma *tm-rstrip-tm*:
assumes $k \geq 2$ **and** $G \geq 4$ **and** $0 < j$ **and** $j < k$
shows *turing-machine k G* (*tm-rstrip z j*)
using *assms tm-right-until-tm tm-left-tm tm-lconst-until-tm tm-cr-tm tm-rstrip-def*
by *simp*

locale *turing-machine-rstrip* =
fixes *z* :: *symbol* **and** *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-right-until j* {□}
definition *tm2* ≡ *tm1* ;; *tm-left j*
definition *tm3* ≡ *tm2* ;; *tm-lconst-until j j* (*UNIV* − {*z*}) □
definition *tm4* ≡ *tm3* ;; *tm-cr j*

lemma *tm4-eq-tm-rstrip*: *tm4* = *tm-rstrip z j*
unfolding *tm4-def tm3-def tm2-def tm1-def tm-rstrip-def* **by** *simp*

context
fixes *tps0* :: *tape list* **and** *zs* :: *symbol list* **and** *k* :: *nat*
assumes *z*: $z > 1$
assumes *zs*: *proper-symbols zs*
assumes *jk*: $0 < j$ $j < k$ *length tps0* = *k*
assumes *tps0*: *tps0* ! *j* = (⌊*zs*⌋, *1*)
begin

definition *tps1* ≡ *tps0*
[*j* := (⌊*zs*⌋, *Suc* (*length zs*))]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *Suc* (*length zs*)
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
proof (*tform tps*: *tps0 tps1-def jk time*: *assms*)
have ∗: *tps0* ! *j* = (⌊*zs*⌋, *1*)
using *tps0 jk* **by** *simp*
show *rneigh* (*tps0* ! *j*) {□} (*length zs*)
using ∗ *zs* **by** (*intro rneighI*) *auto*
qed

definition *tps2* ≡ *tps0*
[*j* := (⌊*zs*⌋, *length zs*)]

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *length zs* + *2*

```
  shows transforms tm2 tps0 ttt tps2
  unfolding tm2-def
  by (tform tps: tps1-def tps2-def jk time: assms)

definition tps3 ≡ tps0
  [j := (⌊rstrip z zs⌋, length (rstrip z zs))]

lemma tm3 [transforms-intros]:
  assumes ttt = length zs + 2 + Suc (length zs − length (rstrip z zs))
  shows transforms tm3 tps0 ttt tps3
  unfolding tm3-def
proof (tform tps: tps2-def tps3-def jk time: assms jk tps2-def)
  let ?n = length zs − length (rstrip z zs)
  have ∗: tps2 ! j = (⌊zs⌋, length zs)
    using tps2-def jk by simp
  show lneigh (tps2 ! j) (UNIV − {z}) ?n
  proof (cases ∃i<length zs. zs ! i ≠ z)
    case True
    then have 1: length (rstrip z zs) > 0
      using rstrip-ex-length by simp
    show ?thesis
    proof (rule lneighI)
      show (tps2 ::: j) (tps2 :#: j − ?n) ∈ UNIV − {z}
        using ∗ 1 contents-inbounds True length-rstrip length-rstrip-le rstrip-def rstrip-ex-last
        by (smt (verit, best) DiffI One-nat-def UNIV-I diff-diff-cancel diff-less fst-conv last-conv-nth
          le-eq-less-or-eq length-greater-0-conv less-Suc-eq-le nth-take singletonD snd-conv)
      have ⋀n'. n' < ?n ⟹ (tps2 ::: j) (tps2 :#: j − n') = z
        using ∗ rstrip-stripped by simp
      then show ⋀n'. n' < ?n ⟹ (tps2 ::: j) (tps2 :#: j − n') ∉ UNIV − {z}
        by simp
    qed
  next
    case False
    then have 1: rstrip z zs = []
      using rstrip-not-ex by simp
    show ?thesis
    proof (rule lneighI)
      show (tps2 ::: j) (tps2 :#: j − ?n) ∈ UNIV − {z}
        using ∗ 1 z by simp
      show ⋀n'. n' < ?n ⟹ (tps2 ::: j) (tps2 :#: j − n') ∉ UNIV − {z}
        using ∗ rstrip-stripped by simp
    qed
  qed

  have lconstplant (⌊zs⌋, length zs) □ ?n = (⌊rstrip z zs⌋, length (rstrip z zs))
    (is ?lhs = -)
  proof −
    have ?lhs = (λi. if length zs − ?n < i ∧ i ≤ length zs then □ else ⌊zs⌋ i, length zs − ?n)
      using lconstplant[of (⌊zs⌋, length zs) 0 ?n] by auto
    moreover have (λi. if length zs − ?n < i ∧ i ≤ length zs then □ else ⌊zs⌋ i) = ⌊rstrip z zs⌋
    proof
      fix i
      consider length zs − ?n < i ∧ i ≤ length zs | i > length zs | i ≤ length (rstrip z zs)
        by linarith
      then show (if length zs − ?n < i ∧ i ≤ length zs then □ else ⌊zs⌋ i) = ⌊rstrip z zs⌋ i
      proof (cases)
        case 1
        then show ?thesis
          by auto
      next
        case 2
        then show ?thesis
          by (metis contents-outofbounds diff-diff-cancel length-rstrip-le less-imp-diff-less)
```

```
      next
        case 3
        then show ?thesis
          using contents-def length-rstrip length-rstrip-le rstrip-def by auto
      qed
    qed
    moreover have length zs − ?n = length (rstrip z zs)
      using diff-diff-cancel length-rstrip-le by simp
    ultimately show ?thesis
      by simp
  qed
  then have lconstplant (tps2 ! j) □ ?n = (⌊rstrip z zs⌋, length (rstrip z zs))
    using tps2-def jk by simp
  then show tps3 = tps2
    [j := tps2 ! j |−| ?n,
     j := lconstplant (tps2 ! j) □ ?n]
    unfolding tps3-def tps2-def by simp
qed

definition tps4 ≡ tps0
  [j := (⌊rstrip z zs⌋, 1)]

lemma tm4:
  assumes ttt = length zs + 2 + Suc (length zs − length (rstrip z zs)) + length (rstrip z zs) + 2
  shows transforms tm4 tps0 ttt tps4
  unfolding tm4-def
proof (tform tps: tps3-def tps4-def jk time: assms tps3-def jk)
  show clean-tape (tps3 ! j)
    using tps3-def jk zs rstrip-def by simp
qed

lemma tm4':
  assumes ttt = 3 ∗ length zs + 5
  shows transforms tm4 tps0 ttt tps4
proof −
  let ?ttt = length zs + 2 + Suc (length zs − length (rstrip z zs)) + length (rstrip z zs) + 2
  have ?ttt = length zs + 5 + (length zs − length (rstrip z zs)) + length (rstrip z zs)
    by simp
  also have ... ≤ length zs + 5 + length zs + length (rstrip z zs)
    by simp
  also have ... ≤ length zs + 5 + length zs + length zs
    using length-rstrip-le by simp
  also have ... = 3 ∗ length zs + 5
    by simp
  finally have ?ttt ≤ 3 ∗ length zs + 5 .
  then show ?thesis
    using assms transforms-monotone tm4 by simp
qed

end

end

lemma transforms-tm-rstripI [transforms-intros]:
  fixes z :: symbol and j :: tapeidx
  fixes tps tps' :: tape list and zs :: symbol list and k :: nat
  assumes z > 1 and 0 < j j < k
    and proper-symbols zs
    and length tps = k
  assumes tps ! j = (⌊zs⌋, 1)
  assumes ttt = 3 ∗ length zs + 5
  assumes tps' = tps[j := (⌊rstrip z zs⌋, 1)]
  shows transforms (tm-rstrip z j) tps ttt tps'
```

proof −
  interpret *loc*: *turing-machine-rstrip z j* **.**
  show *?thesis*
    using *assms loc.tm4′ loc.tps4-def loc.tm4-eq-tm-rstrip* **by** *simp*
qed

2.11.6 Writing arbitrary length sequences of the same symbol

The next Turing machine accepts a number n on tape j_1 and writes the symbol sequence z^n to tape j_2. The symbol z is a parameter of the TM. The TM decrements the number on tape j_1 until it reaches zero.

definition *tm-write-replicate* :: *symbol* ⇒ *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-write-replicate z j1 j2 ≡
    WHILE [] ; *λrs. rs* ! *j1* ≠ □ *DO*
      tm-char j2 z ;;
      tm-decr j1
    DONE ;;
    tm-cr j2

lemma *tm-write-replicate-tm*:
  assumes *0* < *j1* **and** *0* < *j2* **and** *j1* < *k* **and** *j2* < *k* **and** *j1* ≠ *j2* **and** *G* ≥ *4* **and** *z* < *G*
  shows *turing-machine k G* (*tm-write-replicate z j1 j2*)
  unfolding *tm-write-replicate-def*
  using *turing-machine-loop-turing-machine Nil-tm tm-char-tm tm-decr-tm tm-cr-tm assms*
  by *simp*

locale *turing-machine-write-replicate* =
  fixes *j1 j2* :: *tapeidx* **and** *z* :: *symbol*
begin

definition *tm1* ≡ *tm-char j2 z*
definition *tm2* ≡ *tm1* ;; *tm-decr j1*
definition *tmL* ≡ *WHILE* [] ; *λrs. rs* ! *j1* ≠ □ *DO tm2 DONE*
definition *tm3* ≡ *tmL* ;; *tm-cr j2*

lemma *tm3-eq-tm-write-replicate*: *tm3* = *tm-write-replicate z j1 j2*
  using *tm3-def tm2-def tm1-def tm-write-replicate-def tmL-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *k n* :: *nat*
  assumes *jk*: *length tps0* = *k 0* < *j1 0* < *j2 j1* ≠ *j2 j1* < *k j2* < *k*
    and *z*: *1* < *z*
  assumes *tps0*:
    tps0 ! *j1* = (⌊*n*⌋$_N$, *1*)
    tps0 ! *j2* = (⌊[]⌋, *1*)
begin

definition *tpsL* :: *nat* ⇒ *tape list* **where**
  tpsL t ≡ *tps0*
    [*j1* := (⌊*n* − *t*⌋$_N$, *1*),
     j2 := (⌊*replicate t z*⌋, *Suc t*)]

lemma *tpsL0*: *tpsL 0* = *tps0*
  using *tpsL-def tps0 jk* **by** (*metis One-nat-def diff-zero list-update-id replicate-empty*)

definition *tpsL1* :: *nat* ⇒ *tape list* **where**
  tpsL1 t ≡ *tps0*
    [*j1* := (⌊*n* − *t*⌋$_N$, *1*),
     j2 := (⌊*replicate* (*Suc t*) *z*⌋, *Suc* (*Suc t*))]

lemma *tmL1* [*transforms-intros*]: *transforms tm1* (*tpsL t*) *1* (*tpsL1 t*)
  unfolding *tm1-def*
proof (*tform tps*: *tpsL-def tpsL1-def tps0 jk*)
  have *tpsL t* :#: *j2* = *Suc t*

using tpsL1-def jk by (metis length-list-update nth-list-update-eq snd-conv tpsL-def)
  moreover have tpsL t ::: j2 = ⌊replicate t z⌋
    using tpsL1-def jk by (metis fst-conv length-list-update nth-list-update-eq tpsL-def)
  moreover have ⌊replicate t z⌋(Suc t := z) = ⌊replicate (Suc t) z⌋
    using contents-snoc by (metis length-replicate replicate-Suc replicate-append-same)
  ultimately show tpsL1 t = (tpsL t)[j2 := tpsL t ! j2 |:=| z |+| 1]
    unfolding tpsL1-def tpsL-def by simp
qed

lemma tmL2:
  assumes ttt = 9 + 2 * nlength (n − t)
  shows transforms tm2 (tpsL t) ttt (tpsL (Suc t))
  unfolding tm2-def
proof (tform tps: assms tpsL-def tpsL1-def tps0 jk)
  show tpsL (Suc t) = (tpsL1 t)[j1 := (⌊n − t − 1⌋N, 1)]
    unfolding tpsL-def tpsL1-def using jk by (simp add: list-update-swap)
qed

lemma tmL2' [transforms-intros]:
  assumes ttt = 9 + 2 * nlength n
  shows transforms tm2 (tpsL t) ttt (tpsL (Suc t))
proof −
  have 9 + 2 * nlength (n − t) ≤ 9 + 2 * nlength n
    using nlength-mono[of n − t n] by simp
  then show ?thesis
    using assms tmL2 transforms-monotone by blast
qed

lemma tmL [transforms-intros]:
  assumes ttt = n * (11 + 2 * nlength n) + 1
  shows transforms tmL (tpsL 0) ttt (tpsL n)
  unfolding tmL-def
proof (tform)
  let ?t = 9 + 2 * nlength n
  show ⋀i. i < n ⟹ read (tpsL i) ! j1 ≠ □
    using jk tpsL-def read-ncontents-eq-0 by simp
  show ¬ read (tpsL n) ! j1 ≠ □
    using jk tpsL-def read-ncontents-eq-0 by simp
  show n * (9 + 2 * nlength n + 2) + 1 ≤ ttt
    using assms by simp
qed

definition tps3 :: tape list where
  tps3 ≡ tps0
    [j1 := (⌊0⌋N, 1),
     j2 := (⌊replicate n z⌋, 1)]

lemma tm3:
  assumes ttt = n * (12 + 2 * nlength n) + 4
  shows transforms tm3 (tpsL 0) ttt tps3
  unfolding tm3-def
proof (tform tps: z tpsL-def tps3-def tps0 jk)
  have ttt = Suc (n * (11 + 2 * nlength n)) + Suc (Suc (Suc n))
  proof −
    have Suc (n * (11 + 2 * nlength n)) + Suc (Suc (Suc n)) = n * (11 + 2 * nlength n) + 4 + n
      by simp
    also have ... = n * (12 + 2 * nlength n) + 4
      by algebra
    finally have Suc (n * (11 + 2 * nlength n)) + Suc (Suc (Suc n)) = ttt
      using assms by simp
    then show ?thesis
      by simp
  qed

then show *ttt* = *n* ∗ (*11* + *2* ∗ *nlength n*) + *1* + (*tpsL n* :#: *j2* + *2*)
  using *tpsL-def jk* **by** *simp*
qed

lemma *tm3′*:
  assumes *ttt* = *n* ∗ (*12* + *2* ∗ *nlength n*) + *4*
  shows *transforms tm3 tps0 ttt tps3*
  using *tm3 tpsL0 assms* **by** *simp*

end

end

lemma *transforms-tm-write-replicateI* [*transforms-intros*]:
  fixes *j1 j2* :: *tapeidx*
  fixes *tps tps′* :: *tape list* **and** *ttt k n* :: *nat*
  assumes *length tps* = *k 0* < *j1 0* < *j2 j1* ≠ *j2 j1* < *k j2* < *k* **and** *1* < *z*
  assumes
    tps ! *j1* = (⌊*n*⌋$_N$, *1*)
    tps ! *j2* = (⌊[]⌋, *1*)
  assumes *ttt* = *n* ∗ (*12* + *2* ∗ *nlength n*) + *4*
  assumes *tps′* = *tps*
    [*j1* := (⌊*0*⌋$_N$, *1*),
     j2 := (⌊*replicate n z*⌋, *1*)]
  shows *transforms* (*tm-write-replicate z j1 j2*) *tps ttt tps′*
proof −
  interpret *loc*: *turing-machine-write-replicate j1 j2* **.**
  show *?thesis*
    using *assms loc.tm3′ loc.tps3-def loc.tm3-eq-tm-write-replicate* **by** *simp*
qed

2.11.7 Extracting the elements of a pair

In Section 2.1.3 we defined a pairing function for strings. For example, ⟨**𝕀𝕀**, **𝕆𝕆**⟩ is first mapped to **𝕀𝕀**#**𝕆𝕆** and ultimately represented as **𝕆𝕀𝕆𝕀𝕀𝕀𝕆𝕆𝕆𝕆**. A Turing machine that is to compute a function for the argument ⟨**𝕀𝕀**, **𝕆𝕆**⟩ would receive as input the symbols **0101110000**. Typically the TM would then extract the two components **11** and **00**. In this section we devise TMs to do just that.

As it happens, applying the quaternary alphabet decoding function *bindecode* (see Section 2.10) to such a symbol sequence gets us halfway to extracting the elements of the pair. For example, decoding **0101110000** yields **11**♯**00**, and now the TM only has to locate the ♯.

A Turing machine cannot rely on being given a well-formed pair. After decoding, the symbol sequence might have more or fewer than one ♯ symbol or even | symbols. The following functions *first* and *second* are designed to extract the first and second element of a symbol sequence representing a pair, and for other symbol sequences at least allow for an efficient implementation. Implementations will come further down in this section.

definition *first* :: *symbol list* ⇒ *symbol list* **where**
  first ys ≡ *take* (*if* ∃ *i*<*length ys*. *ys* ! *i* ∈ {|, ♯} *then LEAST i*. *i* < *length ys* ∧ *ys* ! *i* ∈ {|, ♯} *else length ys*) *ys*

definition *second* :: *symbol list* ⇒ *symbol list* **where**
  second zs ≡ *drop* (*Suc* (*length* (*first zs*))) *zs*

lemma *firstD*:
  assumes ∃ *i*<*length ys*. *ys* ! *i* ∈ {|, ♯} **and** *m* = (*LEAST i*. *i* < *length ys* ∧ *ys* ! *i* ∈ {|, ♯})
  shows *m* < *length ys* **and** *ys* ! *m* ∈ {|, ♯} **and** ∀ *i*<*m*. *ys* ! *i* ∉ {|, ♯}
  using *LeastI-ex*[*OF assms*(*1*)] *assms*(*2*) **by** *simp-all* (*use less-trans not-less-Least* **in** *blast*)

lemma *firstI*:
  assumes *m* < *length ys* **and** *ys* ! *m* ∈ {|, ♯} **and** ∀ *i*<*m*. *ys* ! *i* ∉ {|, ♯}
  shows (*LEAST i*. *i* < *length ys* ∧ *ys* ! *i* ∈ {|, ♯}) = *m*
  using *assms* **by** (*metis* (*mono-tags*, *lifting*) *LeastI less-linear not-less-Least*)

lemma *length-first-ex*:

assumes $\exists\, i<length\ ys.\ ys\ !\ i \in \{|, \sharp\}$ **and** $m = (LEAST\ i.\ i < length\ ys \wedge ys\ !\ i \in \{|, \sharp\})$
shows *length* (*first ys*) = *m*
proof −
have $m < length\ ys$
using *assms firstD*(*1*) **by** *presburger*
moreover have *first ys* = *take m ys*
using *assms first-def* **by** *simp*
ultimately show *?thesis*
by *simp*
qed

lemma *first-notex*:
assumes $\neg\ (\exists\, i<length\ ys.\ ys\ !\ i \in \{|, \sharp\})$
shows *first ys* = *ys*
using *assms first-def* **by** *auto*

lemma *length-first*: *length* (*first ys*) $\leq$ *length ys*
using *length-first-ex first-notex first-def* **by** *simp*

lemma *length-second-first*: *length* (*second zs*) = *length zs* − *Suc* (*length* (*first zs*))
using *second-def* **by** *simp*

lemma *length-second*: *length* (*second zs*) $\leq$ *length zs*
using *second-def* **by** *simp*

Our next goal is to show that *first* and *second* really extract the first and second element of a pair.

lemma *bindecode-bitenc*:
fixes *x* :: *string*
shows *bindecode* (*string-to-symbols* (*bitenc x*)) = *string-to-symbols x*
proof (*induction x*)
case *Nil*
then show *?case*
using *less-2-cases-iff* **by** *force*
next
case (*Cons a x*)
have *bitenc* (*a* # *x*) = *bitenc* [*a*] @ *bitenc x*
by *simp*
then have *string-to-symbols* (*bitenc* (*a* # *x*)) = *string-to-symbols* (*bitenc* [*a*] @ *bitenc x*)
by *simp*
then have *string-to-symbols* (*bitenc* (*a* # *x*)) = *string-to-symbols* (*bitenc* [*a*]) @ *string-to-symbols* (*bitenc x*)
by *simp*
then have *bindecode* (*string-to-symbols* (*bitenc* (*a* # *x*))) =
bindecode (*string-to-symbols* (*bitenc* [*a*]) @ *string-to-symbols* (*bitenc x*))
by *simp*
also have ... = *bindecode* (*string-to-symbols* (*bitenc* [*a*])) @ *bindecode* (*string-to-symbols* (*bitenc x*))
using *bindecode-append length-bitenc* **by** (*metis* (*no-types*, *lifting*) *dvd-triv-left length-map*)
also have ... = *bindecode* (*string-to-symbols* (*bitenc* [*a*])) @ *string-to-symbols x*
using *Cons* **by** *simp*
also have ... = *string-to-symbols* [*a*] @ *string-to-symbols x*
using *bindecode-def* **by** *simp*
also have ... = *string-to-symbols* ([*a*] @ *x*)
by *simp*
also have ... = *string-to-symbols* (*a* # *x*)
by *simp*
finally show *?case* **.**
qed

lemma *bindecode-string-pair*:
fixes *x u* :: *string*
shows *bindecode* $\langle x;\ u\rangle$ = *string-to-symbols x* @ [$\sharp$] @ *string-to-symbols u*
proof −
have *bindecode* $\langle x;\ u\rangle$ = *bindecode* (*string-to-symbols* (*bitenc x* @ [*True*, *True*] @ *bitenc u*))
using *string-pair-def* **by** *simp*

also have ... = *bindecode*
  (*string-to-symbols* (*bitenc x*) @
   string-to-symbols [𝕀, 𝕀] @
   string-to-symbols (*bitenc u*))
  by *simp*
also have ... = *bindecode* (*string-to-symbols* (*bitenc x*)) @
   bindecode (*string-to-symbols* [𝕀, 𝕀]) @
   bindecode (*string-to-symbols* (*bitenc u*))
proof −
  have *even* (*length* (*string-to-symbols* [*True*, *True*]))
   by *simp*
  moreover have *even* (*length* (*string-to-symbols* (*bitenc y*))) **for** *y*
   by (*simp add*: *length-bitenc*)
  ultimately show *?thesis*
   using *bindecode-append length-bindecode length-bitenc*
   by (*smt* (*verit*) *add-mult-distrib2 add-self-div-2 dvd-triv-left length-append length-map mult-2*)
qed
also have ... = *string-to-symbols x* @ *bindecode* (*string-to-symbols* [𝕀, 𝕀]) @ *string-to-symbols u*
  using *bindecode-bitenc* **by** *simp*
also have ... = *string-to-symbols x* @ [♯] @ *string-to-symbols u*
  using *bindecode-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *first-pair*:
 fixes *ys* :: *symbol list* **and** *x u* :: *string*
 assumes *ys* = *bindecode* ⟨*x*; *u*⟩
 shows *first ys* = *string-to-symbols x*
proof −
 have *ys*: *ys* = *string-to-symbols x* @ [♯] @ *string-to-symbols u*
  using *bindecode-string-pair assms* **by** *simp*
 have *bs*: *bit-symbols* (*string-to-symbols x*)
  by *simp*
 have *ys* ! (*length* (*string-to-symbols x*)) = ♯
  using *ys* **by** (*metis append-Cons nth-append-length*)
 then have *ex*: *ys* ! (*length* (*string-to-symbols x*)) ∈ {|, ♯}
  by *simp*
 have (*LEAST i*. *i* < *length ys* ∧ *ys* ! *i* ∈ {|, ♯}) = *length* (*string-to-symbols x*)
  using *ex ys bs* **by** (*intro firstI*) (*simp-all add*: *nth-append*)
 moreover have *length* (*string-to-symbols x*) < *length ys*
  using *ys* **by** *simp*
 ultimately have *first ys* = *take* (*length* (*string-to-symbols x*)) *ys*
  using *ex first-def* **by** *auto*
 then show *first ys* = *string-to-symbols x*
  using *ys* **by** *simp*
qed

lemma *second-pair*:
 fixes *ys* :: *symbol list* **and** *x u* :: *string*
 assumes *ys* = *bindecode* ⟨*x*; *u*⟩
 shows *second ys* = *string-to-symbols u*
proof −
 have *ys*: *ys* = *string-to-symbols x* @ [♯] @ *string-to-symbols u*
  using *bindecode-string-pair assms* **by** *simp*
 let *?m* = *length* (*string-to-symbols x*)
 have *length* (*first ys*) = *?m*
  using *assms first-pair* **by** *presburger*
 moreover have *drop* (*Suc ?m*) *ys* = *string-to-symbols u*
  using *ys* **by** *simp*
 ultimately have *drop* (*Suc* (*length* (*first ys*))) *ys* = *string-to-symbols u*
  by *simp*
 then show *?thesis*
  using *second-def* **by** *simp*

qed

A Turing machine for extracting the first element

Unlike most other Turing machines, the one in this section is not meant to be reusable, but rather to compute a function, namely the function *first*. For this reason there are no tape index parameters. Instead, the encoded pair is expected on the input tape, and the output is written to the output tape.

lemma *bit-symbols-first*:
  assumes $ys = bindecode\ (string\text{-}to\text{-}symbols\ x)$
  shows *bit-symbols* (*first ys*)
proof ($cases\ \exists\, i<length\ ys.\ ys\ !\ i \in \{|, \sharp\}$)
  case *True*
  define m **where** $m = (LEAST\ i.\ i < length\ ys \wedge ys\ !\ i \in \{|, \sharp\})$
  then have m: $m < length\ ys\ ys\ !\ m \in \{|, \sharp\}\ \forall\, i<m.\ ys\ !\ i \notin \{|, \sharp\}$
    using *firstD*[*OF True*] **by** *blast+*
  have *len*: *length* (*first ys*) = *m*
    using *length-first-ex*[*OF True*] *m-def* **by** *simp*
  have *bit-symbols* (*string-to-symbols x*)
    by *simp*
  then have $\forall\, i<length\ ys.\ ys\ !\ i \in \{2..<6\}$
    using *assms bindecode2345* **by** *simp*
  then have $\forall\, i<m.\ ys\ !\ i \in \{2..<6\}$
    using *m*(*1*) **by** *simp*
  then have $\forall\, i<m.\ ys\ !\ i \in \{2..<4\}$
    using *m*(*3*) **by** *fastforce*
  then show *?thesis*
    using *first-def len* **by** *auto*
next
  case *False*
  then have *1*: $\forall\, i<length\ ys.\ ys\ !\ i \notin \{|, \sharp\}$
    by *simp*
  have *bit-symbols* (*string-to-symbols x*)
    by *simp*
  then have $\forall\, i<length\ ys.\ ys\ !\ i \in \{2..<6\}$
    using *assms bindecode2345* **by** *simp*
  then have $\forall\, i<length\ ys.\ ys\ !\ i \in \{2..<4\}$
    using *1* **by** *fastforce*
  then show *?thesis*
    using *False first-notex* **by** *auto*
qed

definition *tm-first* :: *machine* **where**
  tm-first ≡
    tm-right-many {*0*, *1*, *2*} ;;
    tm-bindecode 0 2 ;;
    tm-cp-until 2 1 {□, |, ♯}

lemma *tm-first-tm*: $G \geq 6 \Longrightarrow k \geq 3 \Longrightarrow$ *turing-machine k G tm-first*
  unfolding *tm-first-def*
  using *tm-cp-until-tm tm-start-tm tm-bindecode-tm tm-right-many-tm*
  by *simp*

locale *turing-machine-fst-pair* =
  fixes *k* :: *nat*
  assumes *k*: $k \geq 3$
begin

definition *tm1* ≡ *tm-right-many* {*0*, *1*, *2*}
definition *tm2* ≡ *tm1* ;; *tm-bindecode 0 2*
definition *tm3* ≡ *tm2* ;; *tm-cp-until 2 1* {□, |, ♯}

lemma *tm3-eq-tm-first*: *tm3* = *tm-first*

```
  using tm1-def tm2-def tm3-def tm-first-def by simp

context
  fixes xs :: symbol list
  assumes bs: bit-symbols xs
begin

definition tps0 ≡ snd (start-config k xs)

lemma lentps [simp]: length tps0 = k
  using tps0-def start-config-length k by simp

lemma tps0-0: tps0 ! 0 = (⌊xs⌋, 0)
  using tps0-def start-config-def contents-def by auto

lemma tps0-gt-0: j > 0 ⟹ j < k ⟹ tps0 ! j = (⌊[]⌋, 0)
  using tps0-def start-config-def contents-def by auto

definition tps1 ≡ tps0
  [0 := (⌊xs⌋, 1),
   1 := (⌊[]⌋, 1),
   2 := (⌊[]⌋, 1)]

lemma tm1 [transforms-intros]: transforms tm1 tps0 1 tps1
  unfolding tm1-def
proof (tform)
  show tps1 = map (λj. if j ∈ {0, 1, 2} then tps0 ! j |+| 1 else tps0 ! j) [0..<length tps0]
    (is - = ?rhs)
  proof (rule nth-equalityI)
    show length tps1 = length ?rhs
      using tps0-def tps1-def by simp
    show tps1 ! j = ?rhs ! j if j < length tps1 for j
      using that tps0-0 tps0-gt-0 tps1-def by simp
  qed
qed

definition tps2 ≡ tps0
  [0 := (⌊xs⌋, 1),
   1 := (⌊[]⌋, 1),
   2 := (⌊bindecode xs⌋, 1)]

lemma tm2 [transforms-intros]:
  assumes ttt = 8 + 3 * length xs
  shows transforms tm2 tps0 ttt tps2
  unfolding tm2-def by (tform tps: bs k tps1-def assms tps2-def)

definition tps3 ≡ tps0
  [0 := (⌊xs⌋, 1),
   1 := (⌊first (bindecode xs)⌋, Suc (length (first (bindecode xs)))),
   2 := (⌊bindecode xs⌋, Suc (length (first (bindecode xs))))]

lemma tm3:
  assumes ttt = 8 + 3 * length xs + Suc (length (first (bindecode xs)))
  shows transforms tm3 tps0 ttt tps3
  unfolding tm3-def
proof (tform tps: k tps2-def time: assms)
  let ?ys = bindecode xs
  have tps2: tps2 ! 2 = (⌊?ys⌋, 1)
    using tps2-def k by simp
  show rneigh (tps2 ! 2) {□, |, ♯} (length (first ?ys))
  proof (cases ∃i<length ?ys. ?ys ! i ∈ {|, ♯})
    case ex5: True
    define m where m = (LEAST i. i < length ?ys ∧ ?ys ! i ∈ {|, ♯})
```

```
    then have m: m = length (first ?ys)
      using length-first-ex ex5 by simp
    show ?thesis
    proof (rule rneighI)
      have ?ys ! m ∈ {|, ♯}
        using firstD m-def ex5 by blast
      then show (tps2 ::: 2) (tps2 :#: 2 + length (first ?ys)) ∈ {□, |, ♯}
        using m tps2 contents-def by simp
      show (tps2 ::: 2) (tps2 :#: 2 + i) ∉ {□, |, ♯} if i < length (first ?ys) for i
      proof −
        have m < length ?ys
          using ex5 firstD(1) length-first-ex m by blast
        then have length (first ?ys) < length ?ys
          using m by simp
        then have i < length ?ys
          using that by simp
        then have ?ys ! i ≠ 0
          using proper-bindecode by fastforce
        moreover have ?ys ! i ∉ {|, ♯}
          using ex5 firstD(3) length-first-ex that by blast
        ultimately show ?thesis
          using Suc-neq-Zero ‹i < length (bindecode xs)› tps2 by simp
      qed
    qed
  next
    case notex5: False
    then have ys: ?ys = first ?ys
      using first-notex by simp
    show ?thesis
    proof (rule rneighI)
      show (tps2 ::: 2) (tps2 :#: 2 + length (first ?ys)) ∈ {□, |, ♯}
        using ys tps2 by simp
      show (tps2 ::: 2) (tps2 :#: 2 + i) ∉ {□, |, ♯} if i < length (first ?ys) for i
        using notex5 that ys proper-bindecode contents-inbounds
        by (metis Suc-leI add-gr-0 diff-Suc-1 fst-conv gr-implies-not0 insert-iff
          plus-1-eq-Suc snd-conv tps2 zero-less-one)
    qed
  qed
  show tps3 = tps2[2 := tps2 ! 2 |+| length (first ?ys), 1 := implant (tps2 ! 2) (tps2 ! 1) (length (first ?ys))]
    (is - = ?tps)
  proof −
    have 0: tps3 ! 0 = ?tps ! 0
      using tps2-def tps3-def by simp
    have 1: tps3 ! 2 = ?tps ! 2
      using tps2-def tps3-def k by simp
    have lentps2: length tps2 > 2
      using k tps2-def by simp
    have implant (tps2 ! 2) (tps2 ! 1) (length (first ?ys)) =
      (⌊first ?ys⌋, Suc (length (first ?ys)))
    proof −
      have len: length (first ?ys) ≤ length ?ys
        using first-def by simp
      have tps2 ! 1 = (⌊[]⌋, 1)
        using tps2-def lentps2 by simp
      then have implant (tps2 ! 2) (tps2 ! 1) (length (first ?ys)) =
          implant (⌊?ys⌋, 1) (⌊[]⌋, 1) (length (first ?ys))
        using tps2 by simp
      also have ... = (⌊take (length (first ?ys)) ?ys⌋, Suc (length (first ?ys)))
        using implant-contents[of 1 length (first ?ys) ?ys []] len by simp
      also have ... = (⌊first ?ys⌋, Suc (length (first ?ys)))
        using first-def using first-notex length-first-ex by presburger
      finally show ?thesis .
    qed
```

```
    moreover have length tps2 > 2
      using k tps2-def by simp
    ultimately show ?thesis
      using 0 1 tps2-def tps3-def tps0-def lentps k tps2
      by (smt (verit) length-list-update list-update-overwrite list-update-swap nth-list-update)
  qed
qed

lemma tm3':
  assumes ttt = 9 + 4 ∗ length xs
  shows transforms tm3 tps0 ttt tps3
proof −
  let ?t = 8 + 3 ∗ length xs + Suc (length (first (bindecode xs)))
  have ?t ≤ 8 + 3 ∗ length xs + Suc (length (bindecode xs))
    using length-first by (meson Suc-le-mono add-le-mono order-refl)
  also have ... ≤ 8 + 3 ∗ length xs + Suc (length xs)
    using length-bindecode by simp
  also have ... = 9 + 3 ∗ length xs + length xs
    by simp
  also have ... = 9 + 4 ∗ length xs
    by simp
  finally have ?t ≤ ttt
    using assms(1) by simp
  moreover have transforms tm3 tps0 ?t tps3
    using tm3 by simp
  ultimately show ?thesis
    using transforms-monotone by simp
qed

end

lemma tm3-computes:
  computes-in-time k tm3 (λx. symbols-to-string (first (bindecode (string-to-symbols x)))) (λn. 9 + 4 ∗ n)
proof −
  define f where f = (λx. symbols-to-string (first (bindecode (string-to-symbols x))))
  define T :: nat ⇒ nat where T = (λn. 9 + 4 ∗ n)
  have computes-in-time k tm3 f T
  proof
    fix x :: string
    let ?xs = string-to-symbols x
    have bs: bit-symbols ?xs
      by simp
    define tps where tps = tps3 ?xs
    have trans: transforms tm3 (tps0 ?xs) (9 + 4 ∗ length ?xs) tps
      using bs tm3' tps-def by blast
    have tps3 ?xs ::: 1 = ⌊first (bindecode ?xs)⌋
      using bs tps3-def k by simp
    moreover have bit-symbols (first (bindecode ?xs))
      using bit-symbols-first by simp
    ultimately have tps3 ?xs ::: 1 = string-to-contents (symbols-to-string (first (bindecode ?xs)))
      using bit-symbols-to-symbols contents-string-to-contents by simp
    then have ∗: tps ::: 1 = string-to-contents (f x)
      using tps-def f-def by auto
    then have transforms tm3 (snd (start-config k (string-to-symbols x))) (T (length x)) tps
      using trans T-def tps0-def by simp
    then show ∃ tps. tps ::: 1 = string-to-contents (f x) ∧
        transforms tm3 (snd (start-config k (string-to-symbols x))) (T (length x)) tps
      using ∗ by auto
  qed
  then show ?thesis
    using f-def T-def by simp
qed
```

end

lemma *tm-first-computes*:
  assumes $k \geq 3$
  shows *computes-in-time*
    k
    tm-first
    (λx. *symbols-to-string* (*first* (*bindecode* (*string-to-symbols* *x*))))
    (λn. $9 + 4 * n$)
proof −
  interpret *loc*: *turing-machine-fst-pair* *k*
    using *turing-machine-fst-pair.intro assms* **by** *simp*
  show *?thesis*
    using *loc.tm3-eq-tm-first loc.tm3-computes* **by** *simp*
qed

A Turing machine for splitting pairs

The next Turing machine expects a proper symbol sequence *zs* on tape j_1 and outputs *first zs* and *second zs* on tapes j_2 and j_3, respectively.

definition *tm-unpair* :: *tapeidx* ⇒ *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-unpair j1 j2 j3 ≡
    tm-cp-until j1 j2 {□, |, ♯} ;;
    tm-right j1 ;;
    tm-cp-until j1 j3 {□} ;;
    tm-cr j1 ;;
    tm-cr j2 ;;
    tm-cr j3

lemma *tm-unpair-tm*:
  assumes $k \geq 2$ **and** $G \geq 4$ **and** $0 < j2$ **and** $0 < j3$ **and** $j1 < k$ $j2 < k$ $j3 < k$
  shows *turing-machine k G* (*tm-unpair j1 j2 j3*)
  using *tm-cp-until-tm tm-right-tm tm-cr-tm assms tm-unpair-def* **by** *simp*

locale *turing-machine-unpair* =
  fixes *j1 j2 j3* :: *tapeidx*
begin

definition *tm1* ≡ *tm-cp-until j1 j2* {□, |, ♯}
definition *tm2* ≡ *tm1* ;; *tm-right j1*
definition *tm3* ≡ *tm2* ;; *tm-cp-until j1 j3* {□}
definition *tm4* ≡ *tm3* ;; *tm-cr j1*
definition *tm5* ≡ *tm4* ;; *tm-cr j2*
definition *tm6* ≡ *tm5* ;; *tm-cr j3*

lemma *tm6-eq-tm-unpair*: *tm6* = *tm-unpair j1 j2 j3*
  unfolding *tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def tm-unpair-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *zs* :: *symbol list*
  assumes *jk*: $0 < j2$ $0 < j3$ $j1 \neq j2$ $j1 \neq j3$ $j2 \neq j3$ $j1 < k$ $j2 < k$ $j3 < k$ *length tps0* = *k*
    and *zs*: *proper-symbols zs*
    and *tps0*:
      tps0 ! *j1* = (⌊*zs*⌋, *1*)
      tps0 ! *j2* = (⌊[]⌋, *1*)
      tps0 ! *j3* = (⌊[]⌋, *1*)
begin

definition *tps1* ≡ *tps0*
  [*j1* := (⌊*zs*⌋, *Suc* (*length* (*first zs*))),
   j2 := (⌊*first zs*⌋, *Suc* (*length* (*first zs*)))]

lemma *tm1* [*transforms-intros*]:

assumes *ttt* = *Suc* (*length* (*first zs*))
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
proof (*tform tps*: *assms tps0 tps1-def jk*)
let *?n* = *length* (*first zs*)
have ∗: *tps0* ! *j1* = (⌊*zs*⌋, *1*)
using *tps0 jk* **by** *simp*
show *rneigh* (*tps0* ! *j1*) {□, |, ♯} (*length* (*first zs*))
proof (*cases* ∃ *i*<*length zs*. *zs* ! *i* ∈ {|, ♯})
case *ex5*: *True*
define *m* **where** *m* = (*LEAST i*. *i* < *length zs* ∧ *zs* ! *i* ∈ {|, ♯})
then have *m*: *m* = *length* (*first zs*)
using *length-first-ex ex5* **by** *simp*
show *?thesis*
proof (*rule rneighI*)
have *zs* ! *m* ∈ {|, ♯}
using *firstD m-def ex5* **by** *blast*
then show (*tps0* ::: *j1*) (*tps0* :#: *j1* + *length* (*first zs*)) ∈ {□, |, ♯}
using *m* ∗ *contents-def* **by** *simp*
show (*tps0* ::: *j1*) (*tps0* :#: *j1* + *i*) ∉ {□, |, ♯} **if** *i* < *length* (*first zs*) **for** *i*
proof −
have *m* < *length zs*
using *ex5 firstD*(*1*) *length-first-ex m* **by** *blast*
then have *length* (*first zs*) < *length zs*
using *m* **by** *simp*
then have *i* < *length zs*
using *that* **by** *simp*
then have *zs* ! *i* ≠ □
using *zs* **by** *fastforce*
moreover have *zs* ! *i* ∉ {|, ♯}
using *ex5 firstD*(*3*) *length-first-ex that* **by** *blast*
ultimately show *?thesis*
using *Suc-neq-Zero* ‹*i* < *length zs*› ∗ **by** *simp*
qed
qed
next
case *notex5*: *False*
then have *ys*: *zs* = *first zs*
using *first-notex* **by** *simp*
show *?thesis*
proof (*rule rneighI*)
show (*tps0* ::: *j1*) (*tps0* :#: *j1* + *length* (*first zs*)) ∈ {□, |, ♯}
using *ys* ∗ **by** *simp*
show (*tps0* ::: *j1*) (*tps0* :#: *j1* + *i*) ∉ {□, |, ♯} **if** *i* < *length* (*first zs*) **for** *i*
using *notex5 that ys proper-bindecode contents-inbounds* ∗ *zs* **by** *auto*
qed
qed

have *1*: *implant* (*tps0* ! *j1*) (*tps0* ! *j2*) *?n* = (⌊*first zs*⌋, *Suc ?n*)
proof −
have *implant* (*tps0* ! *j1*) (*tps0* ! *j2*) *?n* =
(⌊[] @ *take* (*length* (*first zs*)) (*drop* (*1* − *1*) *zs*)⌋,
Suc (*length* []) + *length* (*first zs*))
using *implant-contents*[*of 1 length* (*first zs*) *zs* []] *tps0*(*1*,*2*)
by (*metis* (*mono-tags*, *lifting*) *add.right-neutral diff-Suc-1 le-eq-less-or-eq*
firstD(*1*) *first-notex length-first-ex less-one list.size*(*3*) *plus-1-eq-Suc*)
then have *implant* (*tps0* ! *j1*) (*tps0* ! *j2*) *?n* = (⌊*take ?n zs*⌋, *Suc ?n*)
by *simp*
then show *implant* (*tps0* ! *j1*) (*tps0* ! *j2*) *?n* = (⌊*first zs*⌋, *Suc ?n*)
using *first-def length-first-ex* **by** *auto*
qed
have *2*: *tps0* ! *j1* |+| *?n* = (⌊*zs*⌋, *Suc ?n*)
using *tps0 jk* **by** *simp*

show *tps1* = *tps0*
  [*j1* := *tps0* ! *j1* |+| *?n*,
   j2 := *implant* (*tps0* ! *j1*) (*tps0* ! *j2*) *?n*]
 unfolding *tps1-def* **using** *jk 1 2* **by** *simp*
qed

definition *tps2* ≡ *tps0*
 [*j1* := (⌊*zs*⌋, *length* (*first zs*) + *2*),
  j2 := (⌊*first zs*⌋, *Suc* (*length* (*first zs*)))]

lemma *tm2* [*transforms-intros*]:
 assumes *ttt* = *length* (*first zs*) + *2*
 shows *transforms tm2 tps0 ttt tps2*
 unfolding *tm2-def*
proof (*tform tps*: *tps1-def jk tps2-def time*: *assms*)
 have *tps1* ! *j1* |+| *1* = (⌊*zs*⌋, *length* (*first zs*) + *2*)
  using *tps1-def jk* **by** *simp*
 then show *tps2* = *tps1*[*j1* := *tps1* ! *j1* |+| *1*]
  unfolding *tps2-def tps1-def* **using** *jk* **by** (*simp add*: *list-update-swap*)
qed

definition *tps3* ≡ *tps0*
 [*j1* := (⌊*zs*⌋, *length* (*first zs*) + *2* + (*length zs* − *Suc* (*length* (*first zs*)))),
  j2 := (⌊*first zs*⌋, *Suc* (*length* (*first zs*))),
  j3 := (⌊*second zs*⌋, *Suc* (*length* (*second zs*)))]

lemma *tm3* [*transforms-intros*]:
 assumes *ttt* = *length* (*first zs*) + *2* + *Suc* (*length zs* − *Suc* (*length* (*first zs*)))
 shows *transforms tm3 tps0 ttt tps3*
 unfolding *tm3-def*
proof (*tform tps*: *assms tps2-def tps3-def jk*)
 let *?ll* = *length* (*first zs*)
 let *?n* = *length zs* − *Suc ?ll*
 have *at-j1*: *tps2* ! *j1* = (⌊*zs*⌋, *length* (*first zs*) + *2*)
  using *tps2-def jk* **by** *simp*
 show *rneigh* (*tps2* ! *j1*) {*0*} *?n*
 proof (*rule rneighI*)
  show (*tps2* ::: *j1*) (*tps2* :#: *j1* + (*length zs* − *Suc ?ll*)) ∈ {*0*}
   using *at-j1* **by** *simp*
  show (*tps2* ::: *j1*) (*tps2* :#: *j1* + *m*) ∉ {*0*} **if** *m* < *length zs* − *Suc ?ll* **for** *m*
  proof −
   have ∗: (*tps2* ::: *j1*) (*tps2* :#: *j1* + *m*) = ⌊*zs*⌋ (*?ll* + *2* + *m*)
    using *at-j1* **by** *simp*
   have *Suc ?ll* < *length zs*
    using *that* **by** *simp*
   then have *?ll* + *2* + *m* ≤ *Suc* (*length zs*)
    using *that* **by** *simp*
   then have ⌊*zs*⌋ (*?ll* + *2* + *m*) = *zs* ! (*?ll* + *1* + *m*)
    using *that* **by** *simp*
   then have ⌊*zs*⌋ (*?ll* + *2* + *m*) > *0*
    using *zs that* **by** (*metis add.commute gr0I less-diff-conv not-add-less2 plus-1-eq-Suc*)
   then show *?thesis*
    using ∗ **by** *simp*
  qed
 qed

 have *1*: *implant* (*tps2* ! *j1*) (*tps2* ! *j3*) *?n* = (⌊*second zs*⌋, *Suc* (*length* (*second zs*)))
 proof (*cases Suc ?ll* ≤ *length zs*)
  case *True*
  have *implant* (*tps2* ! *j1*) (*tps2* ! *j3*) *?n* = *implant* (⌊*zs*⌋, *?ll* + *2*) (⌊[]⌋, *1*) *?n*
   using *tps2-def jk* **by** (*metis at-j1 nth-list-update-neq′ tps0*(*3*))
  also have ... = (⌊*take ?n* (*drop* (*Suc ?ll*) *zs*)⌋, *Suc ?n*)
   using *True implant-contents*

by (*metis* (*no-types*, *lifting*) *One-nat-def add.commute add-2-eq-Suc′ append.simps*(*1*) *diff-Suc-1 dual-order.refl le-add-diff-inverse2 list.size*(*3*) *plus-1-eq-Suc zero-less-Suc*)
also have ... = (⌊*take* (*length* (*second zs*)) (*drop* (*Suc ?ll*) *zs*)⌋, *Suc* (*length* (*second zs*)))
using *length-second-first* **by** *simp*
also have ... = (⌊*second zs*⌋, *Suc* (*length* (*second zs*)))
using *second-def* **by** *simp*
finally show *?thesis* **.**
next
case *False*
then have *?n* = *0*
by *simp*
then have *implant* (*tps2* ! *j1*) (*tps2* ! *j3*) *?n* = *implant* (⌊*zs*⌋, *?ll* + *2*) (⌊[]⌋, *1*) *0*
using *tps2-def jk* **by** (*metis at-j1 nth-list-update-neq′ tps0*(*3*))
then have *implant* (*tps2* ! *j1*) (*tps2* ! *j3*) *?n* = (⌊[]⌋, *1*)
using *transplant-0* **by** *simp*
moreover have *second zs* = []
using *False second-def* **by** *simp*
ultimately show *?thesis*
by *simp*
qed

show *tps3* = *tps2*
[*j1* := *tps2* ! *j1* |+| *?n*,
j3 := *implant* (*tps2* ! *j1*) (*tps2* ! *j3*) *?n*]
using *tps3-def tps2-def* **using** *1 jk at-j1* **by** (*simp add*: *list-update-swap*[*of j1*])
qed

definition *tps4* ≡ *tps0*
[*j1* := (⌊*zs*⌋, *1*),
j2 := (⌊*first zs*⌋, *Suc* (*length* (*first zs*))),
j3 := (⌊*second zs*⌋, *Suc* (*length* (*second zs*)))]

lemma *tm4*:
assumes *ttt* = *2* ∗ *length* (*first zs*) + *7* + *2* ∗ (*length zs* − *Suc* (*length* (*first zs*)))
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform tps*: *assms tps3-def tps4-def jk zs*)
have *tps3* ! *j1* |#=| *1* = (⌊*zs*⌋, *1*)
using *tps3-def jk* **by** *simp*
then show *tps4* = *tps3*[*j1* := *tps3* ! *j1* |#=| *1*]
unfolding *tps4-def tps3-def* **using** *jk* **by** (*simp add*: *list-update-swap*)
qed

lemma *tm4′* [*transforms-intros*]:
assumes *ttt* = *4* ∗ *length zs* + *7*
shows *transforms tm4 tps0 ttt tps4*
proof −
have *2* ∗ *length* (*first zs*) + *7* + *2* ∗ (*length zs* − *Suc* (*length* (*first zs*))) ≤ *2* ∗ *length* (*first zs*) + *7* + *2* ∗ *length zs*
by *simp*
also have ... ≤ *2* ∗ *length zs* + *7* + *2* ∗ *length zs*
using *length-first* **by** *simp*
also have ... = *ttt*
using *assms* **by** *simp*
finally have *2* ∗ *length* (*first zs*) + *7* + *2* ∗ (*length zs* − *Suc* (*length* (*first zs*))) ≤ *ttt* **.**
then show *?thesis*
using *assms tm4 transforms-monotone* **by** *simp*
qed

definition *tps5* ≡ *tps0*
[*j1* := (⌊*zs*⌋, *1*),
j2 := (⌊*first zs*⌋, *1*),
j3 := (⌊*second zs*⌋, *Suc* (*length* (*second zs*)))]

```
lemma tm5 [transforms-intros]:
  assumes ttt = 4 ∗ length zs + 9 + Suc (length (first zs))
  shows transforms tm5 tps0 ttt tps5
  unfolding tm5-def
proof (tform tps: assms tps4-def tps5-def jk)
  show clean-tape (tps4 ! j2)
    using zs first-def tps4-def jk by simp
  have tps4 ! j2 |#=| 1 = (⌊first zs⌋, 1)
    using tps4-def jk by simp
  then show tps5 = tps4[j2 := tps4 ! j2 |#=| 1]
    unfolding tps5-def tps4-def using jk by (simp add: list-update-swap)
qed

definition tps6 ≡ tps0
  [j1 := (⌊zs⌋, 1),
   j2 := (⌊first zs⌋, 1),
   j3 := (⌊second zs⌋, 1)]

lemma tm6:
  assumes ttt = 4 ∗ length zs + 11 + Suc (length (first zs)) + Suc (length (second zs))
  shows transforms tm6 tps0 ttt tps6
  unfolding tm6-def
proof (tform tps: assms tps5-def tps6-def jk)
  show clean-tape (tps5 ! j3)
    using zs second-def tps5-def jk by simp
qed

definition tps6' ≡ tps0
  [j2 := (⌊first zs⌋, 1),
   j3 := (⌊second zs⌋, 1)]

lemma tps6': tps6' = tps6
  using tps6-def tps6'-def list-update-id tps0(1) by metis

lemma tm6':
  assumes ttt = 6 ∗ length zs + 13
  shows transforms tm6 tps0 ttt tps6'
proof −
  have 4 ∗ length zs + 11 + Suc (length (first zs)) + Suc (length (second zs)) ≤
      4 ∗ length zs + 13 + length zs + length (second zs)
    using length-first by simp
  also have ... ≤ 6 ∗ length zs + 13
    using length-second by simp
  finally have 4 ∗ length zs + 11 + Suc (length (first zs)) + Suc (length (second zs)) ≤ ttt
    using assms by simp
  then show ?thesis
    using tm6 tps6' transforms-monotone by simp
qed

end

end

lemma transforms-tm-unpairI [transforms-intros]:
  fixes j1 j2 j3 :: tapeidx
  fixes tps tps' :: tape list and k :: nat and zs :: symbol list
  assumes 0 < j2 0 < j3 j1 ≠ j2 j1 ≠ j3 j2 ≠ j3 j1 < k j2 < k j3 < k
    and length tps = k
    and proper-symbols zs
  assumes
    tps ! j1 = (⌊zs⌋, 1)
    tps ! j2 = (⌊[]⌋, 1)
```

```
    tps ! j3 = (⌊[]⌋, 1)
  assumes ttt = 6 * length zs + 13
  assumes tps' = tps
    [j2 := (⌊first zs⌋, 1),
     j3 := (⌊second zs⌋, 1)]
  shows transforms (tm-unpair j1 j2 j3) tps ttt tps'
proof -
  interpret loc: turing-machine-unpair j1 j2 j3 .
  show ?thesis
    using assms loc.tps6'-def loc.tm6' loc.tm6-eq-tm-unpair by metis
qed

end
```

2.12 Well-formedness of lists

```
theory Wellformed
  imports Symbol-Ops Lists-Lists
begin
```

In the representations introduced in Section 2.8 and Section 2.9, not every symbol sequence over **01**| represents a list of numbers, and not every symbol sequence over **01**|♯ represents a list of lists of numbers. In this section we prove criteria for symbol sequences to represent such lists and devise Turing machines to check these criteria efficiently.

2.12.1 A criterion for well-formed lists

From the definition of *numlist* it is easy to see that a symbol sequence representing a list of numbers is either empty or not, and that in the latter case it ends with a | symbol. Moreover it can only contain the symbols **01**| and cannot contain the symbol sequence **0**| because canonical number representations cannot end in **0**. That these properties are not only necessary but also sufficient for the symbol sequence to represent a list of numbers is shown in this section.

A symbol sequence is well-formed if it represents a list of numbers.

```
definition numlist-wf :: symbol list ⇒ bool where
  numlist-wf zs ≡ ∃ ns. numlist ns = zs

lemma numlist-wf-append:
  assumes numlist-wf xs and numlist-wf ys
  shows numlist-wf (xs @ ys)
proof -
  obtain ms ns where numlist ms = xs and numlist ns = ys
    using assms numlist-wf-def by auto
  then have numlist (ms @ ns) = xs @ ys
    using numlist-append by simp
  then show ?thesis
    using numlist-wf-def by auto
qed

lemma numlist-wf-canonical:
  assumes canonical xs
  shows numlist-wf (xs @ [|])
proof -
  obtain n where canrepr n = xs
    using assms canreprI by blast
  then have numlist [n] = xs @ [|]
    using numlist-def by simp
  then show ?thesis
    using numlist-wf-def by auto
qed
```

Well-formed symbol sequences can be unambiguously decoded to lists of numbers.

definition *zs-numlist* :: *symbol list* ⇒ *nat list* **where**
  zs-numlist zs ≡ *THE ns. numlist ns* = *zs*

lemma *zs-numlist-ex1*:
  assumes *numlist-wf zs*
  shows ∃!*ns. numlist ns* = *zs*
  using *assms numlist-wf-def numlist-inj* **by** *blast*

lemma *numlist-zs-numlist*:
  assumes *numlist-wf zs*
  shows *numlist* (*zs-numlist zs*) = *zs*
  using *assms zs-numlist-def zs-numlist-ex1* **by** (*smt* (*verit*, *del-insts*) *the-equality*)

Count the number of occurrences of an element in a list:

fun *count* :: *nat list* ⇒ *nat* ⇒ *nat* **where**
  count [] *z* = *0* |
  count (*x* # *xs*) *z* = (*if x* = *z then 1 else 0*) + *count xs z*

lemma *count-append*: *count* (*xs* @ *ys*) *z* = *count xs z* + *count ys z*
  by (*induction xs*) *simp-all*

lemma *count-0*: *count xs z* = *0* ⟷ (∀ *x*∈*set xs. x* ≠ *z*)
proof
  show *count xs z* = *0* ⟹ ∀ *x*∈*set xs. x* ≠ *z*
    by (*induction xs*) *auto*
  show ∀ *x*∈*set xs. x* ≠ *z* ⟹ *count xs z* = *0*
    by (*induction xs*) *auto*
qed

lemma *count-gr-0-take*:
  assumes *count xs z* > *0*
  shows ∃ *j*.
    j < *length xs* ∧
    xs ! *j* = *z* ∧
    (∀ *i*<*j*. *xs* ! *i* ≠ *z*) ∧
    count (*take* (*Suc j*) *xs*) *z* = *1* ∧
    count (*drop* (*Suc j*) *xs*) *z* = *count xs z* − *1*
proof −
  let *?P* = λ*i*. *i* < *length xs* ∧ *xs* ! *i* = *z*
  have *ex*: ∃ *i*. *?P i*
    using *assms*(*1*) *count-0* **by** (*metis bot-nat-0.not-eq-extremum in-set-conv-nth*)
  define *j* **where** *j* = *Least ?P*
  have *1*: *j* < *length xs*
    using *j-def ex* **by** (*metis* (*mono-tags*, *lifting*) *LeastI*)
  moreover have *2*: *xs* ! *j* = *z*
    using *j-def ex* **by** (*metis* (*mono-tags*, *lifting*) *LeastI*)
  moreover have *3*: ∀ *i*<*j*. *xs* ! *i* ≠ *z*
    using *j-def ex 1 not-less-Least order-less-trans* **by** *blast*
  moreover have *4*: *count* (*take* (*Suc j*) *xs*) *z* = *1*
  proof −
    have ∀ *x*∈*set* (*take j xs*). *x* ≠ *z*
      using *3 1* **by** (*metis in-set-conv-nth length-take less-imp-le-nat min-absorb2 nth-take*)
    then have *count* (*take j xs*) *z* = *0*
      using *count-0* **by** *simp*
    moreover have *count* [*xs* ! *j*] *z* = *1*
      using *2* **by** *simp*
    moreover have *take* (*Suc j*) *xs* = *take j xs* @ [*xs* ! *j*]
      using *1 take-Suc-conv-app-nth* **by** *auto*
    ultimately show *count* (*take* (*Suc j*) *xs*) *z* = *1*
      using *count-append* **by** *simp*
  qed
  moreover have *count* (*drop* (*Suc j*) *xs*) *z* = *count xs z* − *1*
  proof −

have *xs* = *take* (*Suc j*) *xs* @ *drop* (*Suc j*) *xs*
using *1* **by** *simp*
then show *?thesis*
using *count-append 4* **by** (*metis add-diff-cancel-left'*)
qed
ultimately show *?thesis*
by *auto*
qed

definition *has2* :: *symbol list* ⇒ *symbol* ⇒ *symbol* ⇒ *bool* **where**
has2 xs y z ≡ ∃ *i*<*length xs* − *1*. *xs* ! *i* = *y* ∧ *xs* ! (*Suc i*) = *z*

lemma *not-has2-take*:
assumes ¬ *has2 xs y z*
shows ¬ *has2* (*take m xs*) *y z*
proof (*rule ccontr*)
let *?ys* = *take m xs*
assume ¬ ¬ *has2 ?ys y z*
then have *has2 ?ys y z*
by *simp*
then have *has2 xs y z*
using *has2-def* **by** *fastforce*
then show *False*
using *assms* **by** *simp*
qed

lemma *not-has2-drop*:
assumes ¬ *has2 xs y z*
shows ¬ *has2* (*drop m xs*) *y z*
proof (*rule ccontr*)
let *?ys* = *drop m xs*
assume ¬ ¬ *has2 ?ys y z*
then have *has2 ?ys y z*
by *simp*
then have *has2 xs y z*
using *has2-def* **by** *fastforce*
then show *False*
using *assms* **by** *simp*
qed

lemma *numlist-wf-has2*:
assumes *proper-symbols xs symbols-lt 5 xs* ¬ *has2 xs* **0** | *xs* ≠ [] ⟶ *last xs* = |
shows *numlist-wf xs*
using *assms*
proof (*induction count xs* | *arbitrary*: *xs*)
case *0*
then have *xs* = []
using *count-0* **by** *simp*
then show *?case*
using *numlist-wf-def numlist-Nil* **by** *blast*
next
case (*Suc n*)
then obtain *j* :: *nat* **where** *j*:
j < *length xs*
xs ! *j* = |
∀ *i*<*j*. *xs* ! *i* ≠ |
count (*take* (*Suc j*) *xs*) | = *1*
count (*drop* (*Suc j*) *xs*) | = *count xs* | − *1*
by (*metis count-gr-0-take zero-less-Suc*)
then have *xs* ≠ []
by *auto*
then have *last xs* = |
using *Suc.prems* **by** *simp*

let *?ys* = *drop* (*Suc j*) *xs*
have *count ?ys* | = *n*
  using *j*(*5*) *Suc* **by** *simp*
moreover have *proper-symbols ?ys*
  using *Suc.prems* **by** *simp*
moreover have *symbols-lt 5 ?ys*
  using *Suc.prems* **by** *simp*
moreover have ¬ *has2 ?ys* **0** |
  using *not-has2-drop Suc.prems*(*3*) **by** *simp*
moreover have *?ys* ≠ [] ⟶ *last ?ys* = |
  using *j* **by** (*simp add*: ‹*last xs* = |›)
ultimately have *wf-ys*: *numlist-wf ?ys*
  using *Suc* **by** *simp*

let *?zs* = *take j xs*
have *canonical ?zs*
proof −
  have *?zs* ! *i* ≥ **0 if** *i* < *length ?zs* **for** *i*
    using *that Suc.prems*(*1*) *j* **by** (*metis One-nat-def Suc-1 Suc-leI length-take min-less-iff-conj nth-take*)
  moreover have *?zs* ! *i* ≤ **1 if** *i* < *length ?zs* **for** *i*
  proof −
    have *?zs* ! *i* < |
      using *that Suc.prems*(*1*,*2*) *j*
      by (*metis eval-nat-numeral*(*3*) *length-take less-Suc-eq-le min-less-iff-conj nat-less-le nth-take*)
    then show *?thesis*
      by *simp*
  qed
  ultimately have *bit-symbols ?zs*
    by *fastforce*
  moreover have *?zs* = [] ∨ *last ?zs* = **1**
  proof (*cases ?zs* = [])
    case *True*
    then show *?thesis*
      by *simp*
  next
    case *False*
    then have *last ?zs* = *?zs* ! (*j* − *1*)
      by (*metis add-diff-inverse-nat j*(*1*) *last-length length-take less-imp-le-nat less-one min-absorb2 plus-1-eq-Suc take-eq-Nil*)
    then have *last*: *last ?zs* = *xs* ! (*j* − *1*)
      using *False* **by** *simp*
    have *xs* ! (*j* − *1*) ≠ |
      using *j*(*3*) *False* **by** *simp*
    moreover have *xs* ! (*j* − *1*) < ♯
      using *Suc.prems*(*2*) *j*(*1*) **by** *simp*
    moreover have *xs* ! (*j* − *1*) ≥ **0**
      using *Suc.prems*(*1*) *j*(*1*) **by** (*metis One-nat-def Suc-1 Suc-leI less-imp-diff-less*)
    moreover have *xs* ! (*j* − *1*) ≠ **0**
    proof (*rule ccontr*)
      assume ¬ *xs* ! (*j* − *1*) ≠ **0**
      then have *xs* ! (*j* − *1*) = **0**
        by *simp*
      moreover have *xs* ! *j* = |
        using *j* **by** *simp*
      ultimately have *has2 xs* **0** |
        using *has2-def j False*
          by (*metis* (*no-types*, *lifting*) *Nat.lessE add-diff-cancel-left′ less-Suc-eq-0-disj not-less-eq plus-1-eq-Suc take-eq-Nil*)
      then show *False*
        using *Suc.prems*(*3*) **by** *simp*
    qed
    ultimately have *xs* ! (*j* − *1*) = **1**

```
      by simp
    then have last ?zs = 1
      using last by simp
    then show ?thesis
      by simp
  qed
  ultimately show canonical ?zs
    using canonical-def by simp
qed

let ?ts = take (Suc j) xs
have ?ts = ?zs @ [|]
  using j by (metis take-Suc-conv-app-nth)
then have numlist-wf ?ts
  using numlist-wf-canonical ‹canonical ?zs› by simp
moreover have xs = ?ts @ ?ys
  by simp
ultimately show numlist-wf xs
  using wf-ys numlist-wf-append by fastforce
qed

lemma last-numlist-4: numlist ns ≠ [] ⟹ last (numlist ns) = |
proof (induction ns)
  case Nil
  then show ?case
    using numlist-def by simp
next
  case (Cons n ns)
  then show ?case
    using numlist-def by (cases numlist ns = []) simp-all
qed

lemma numlist-not-has2:
  assumes i < length (numlist ns) − 1 and numlist ns ! i = 0
  shows numlist ns ! (Suc i) ≠ |
  using assms
proof (induction ns arbitrary: i)
  case Nil
  then show ?case
    by (simp add: numlist-Nil)
next
  case (Cons n ns)
  show numlist (n # ns) ! (Suc i) ≠ |
  proof (cases i < length (numlist [n]))
    case True
    have numlist (n # ns) ! i = (numlist [n] @ numlist ns) ! i
      using numlist-def by simp
    then have numlist (n # ns) ! i = numlist [n] ! i
      using True by (simp add: nth-append)
    then have numlist (n # ns) ! i = (canrepr n @ [|]) ! i
      using numlist-def by simp
    moreover have numlist (n # ns) ! i = 0
      using Cons by simp
    ultimately have (canrepr n @ [|]) ! i = 0
      by simp
    moreover have (canrepr n @ [|]) ! (length (canrepr n @ [|]) − 1) = |
      by simp
    ultimately have i ≠ length (canrepr n @ [|]) − 1
      by auto
    then have *: i ≠ length (numlist [n]) − 1
      using numlist-def by simp

    have 3: canrepr n ! j = numlist (n # ns) ! j if j < nlength n for j
```

proof −
  have *j*: *j* < *length* (*numlist* [*n*])
    using *that numlist-def* **by** *simp*
  have *numlist* (*n* # *ns*) ! *j* = (*numlist* [*n*] @ *numlist ns*) ! *j*
    using *numlist-def* **by** *simp*
  then have *numlist* (*n* # *ns*) ! *j* = *numlist* [*n*] ! *j*
    using *j* **by** (*simp add*: *nth-append*)
  then have *numlist* (*n* # *ns*) ! *j* = (*canrepr n* @ [|]) ! *j*
    using *numlist-def* **by** *simp*
  then show *?thesis*
    by (*simp add*: *nth-append that*)
qed

have *neq0*: *n* ≠ *0*
proof −
  have *length* (*numlist* [*0*]) = *1*
    using *numlist-def* **by** *simp*
  then show *?thesis*
    using ∗ *True* **by** (*metis diff-self-eq-0 less-one*)
qed
then have *i* < *length* (*numlist* [*n*]) − *1*
  using ∗ *True* **by** *simp*
then have *i* < *length* (*canrepr n* @ [|]) − *1*
  using *numlist-def* **by** *simp*
then have *i* < *length* (*canrepr n*)
  by *simp*
then have *canrepr n* ! *i* = **0**
  by (*metis* ‹(*canrepr n* @ [|]) ! *i* = **0**› *nth-append*)
moreover have *last* (*canrepr n*) ≠ **0**
  using *canonical-canrepr canonical-def*
  by (*metis neq0 length-0-conv n-not-Suc-n nlength-0 numeral-2-eq-2 numeral-3-eq-3*)
ultimately have *i* ≠ *nlength n* − *1*
  by (*metis* ‹*i* < *nlength n*› *last-conv-nth less-zeroE list.size*(*3*))
then have *i* < *nlength n* − *1*
  using ‹*i* < *nlength n*› **by** *linarith*
then have *Suc i* < *nlength n*
  by *simp*
then have *canrepr n* ! *Suc i* ≤ **1**
  using *bit-symbols-canrepr* **by** *fastforce*
moreover have *canrepr n* ! *Suc i* = *numlist* (*n* # *ns*) ! *Suc i*
  using *3* ‹*Suc i* < *nlength n*› **by** *blast*
ultimately show *?thesis*
  by *simp*
next
case *False*
let *?i* = *i* − *length* (*numlist* [*n*])
have *numlist* (*n* # *ns*) ! *i* = (*numlist* [*n*] @ *numlist ns*) ! *i*
  using *numlist-def* **by** *simp*
then have *numlist* (*n* # *ns*) ! *i* = *numlist ns* ! *?i*
  using *False* **by** (*simp add*: *nth-append*)
then have *numlist ns* ! *?i* = **0**
  using *Cons* **by** *simp*
moreover have *?i* < *length* (*numlist ns*) − *1*
proof −
  have *length* (*numlist* (*n* # *ns*)) = *length* (*numlist* [*n*]) + *length* (*numlist ns*)
    using *numlist-def* **by** *simp*
  then show *?thesis*
    using *False Cons* **by** *simp*
qed
ultimately have *numlist ns* ! *Suc ?i* ≠ |
  using *Cons* **by** *simp*
moreover have *numlist* (*n* # *ns*) ! *Suc i* = *numlist ns* ! *Suc ?i*
  using *False numlist-append*

by (*smt* (*verit*, *del-insts*) *Suc-diff-Suc Suc-lessD append-Cons append-Nil diff-Suc-Suc not-less-eq nth-append*)
ultimately show *?thesis*
by *simp*
qed
qed

lemma *numlist-wf-has2'*:
assumes *numlist-wf xs*
shows *proper-symbols-lt 5 xs* ∧ ¬ *has2 xs* **0** | ∧ (*xs* ≠ [] ⟶ *last xs* = |)
proof −
obtain *ns* **where** *ns*: *numlist ns* = *xs*
using *numlist-wf-def assms* **by** *auto*
have *proper-symbols xs*
using *proper-symbols-numlist ns* **by** *auto*
moreover have *symbols-lt 5 xs*
using *ns numlist-234*
by (*smt* (*verit*, *best*) *One-nat-def Suc-1 eval-nat-numeral*(*3*) *in-mono insertE less-Suc-eq-le linorder-le-less-linear nle-le not-less0 nth-mem numeral-less-iff semiring-norm*(*76*) *semiring-norm*(*89*) *semiring-norm*(*90*) *singletonD*)
moreover have ¬ *has2 xs* **0** |
using *numlist-not-has2 ns has2-def* **by** *auto*
moreover have *xs* ≠ [] ⟶ *last xs* = |
using *last-numlist-4 ns* **by** *auto*
ultimately show *?thesis*
by *simp*
qed

lemma *numlist-wf-iff*:
numlist-wf xs ⟷ *proper-symbols-lt 5 xs* ∧ ¬ *has2 xs* **0** | ∧ (*xs* ≠ [] ⟶ *last xs* = |)
using *numlist-wf-has2 numlist-wf-has2'* **by** *auto*

2.12.2 A criterion for well-formed lists of lists

The criterion for lists of lists of numbers is similar to the one for lists of numbers. A non-empty symbol sequence must end in ♯. All symbols must be from **01**|♯ and the sequences **0**|, **0**♯, and **1**♯ are forbidden. A symbol sequence is well-formed if it represents a list of lists of numbers.

definition *numlistlist-wf* :: *symbol list* ⇒ *bool* **where**
numlistlist-wf zs ≡ ∃ *nss*. *numlistlist nss* = *zs*

lemma *numlistlist-wf-append*:
assumes *numlistlist-wf xs* **and** *numlistlist-wf ys*
shows *numlistlist-wf* (*xs* @ *ys*)
proof −
obtain *ms ns* **where** *numlistlist ms* = *xs* **and** *numlistlist ns* = *ys*
using *assms numlistlist-wf-def* **by** *auto*
then have *numlistlist* (*ms* @ *ns*) = *xs* @ *ys*
using *numlistlist-append* **by** *simp*
then show *?thesis*
using *numlistlist-wf-def* **by** *auto*
qed

lemma *numlistlist-wf-numlist-wf*:
assumes *numlist-wf xs*
shows *numlistlist-wf* (*xs* @ [♯])
proof −
obtain *ns* **where** *numlist ns* = *xs*
using *assms numlist-wf-def* **by** *auto*
then have *numlistlist* [*ns*] = *xs* @ [♯]
using *numlistlist-def* **by** *simp*
then show *?thesis*
using *numlistlist-wf-def* **by** *auto*
qed

```
lemma numlistlist-wf-has2:
  assumes proper-symbols xs symbols-lt 6 xs xs ≠ [] ⟶ last xs = ♯
    and ¬ has2 xs 0 |
    and ¬ has2 xs 0 ♯
    and ¬ has2 xs 1 ♯
  shows numlistlist-wf xs
  using assms
proof (induction count xs ♯ arbitrary: xs)
  case 0
  then have xs = []
    using count-0 by simp
  then show ?case
    using numlistlist-wf-def numlistlist-Nil by auto
next
  case (Suc n)
  then obtain j :: nat where j:
    j < length xs
    xs ! j = ♯
    ∀ i<j. xs ! i ≠ ♯
    count (take (Suc j) xs) ♯ = 1
    count (drop (Suc j) xs) ♯ = count xs ♯ − 1
    by (metis count-gr-0-take zero-less-Suc)
  then have xs ≠ []
    by auto
  then have last xs = ♯
    using Suc.prems by simp
  let ?ys = drop (Suc j) xs
  have count ?ys ♯ = n
    using j(5) Suc by simp
  moreover have proper-symbols ?ys
    using Suc.prems(1) by simp
  moreover have symbols-lt 6 ?ys
    using Suc.prems(2) by simp
  moreover have ?ys ≠ [] ⟶ last ?ys = ♯
    using j by (simp add: ‹last xs = ♯›)
  moreover have ¬ has2 ?ys 0 |
    using not-has2-drop Suc.prems(4) by simp
  moreover have ¬ has2 ?ys 0 ♯
    using not-has2-drop Suc.prems(5) by simp
  moreover have ¬ has2 ?ys 1 ♯
    using not-has2-drop Suc.prems(6) by simp
  ultimately have wf-ys: numlistlist-wf ?ys
    using Suc by simp

  let ?zs = take j xs
  have len: length ?zs = j
    using j(1) by simp
  have numlist-wf ?zs
  proof −
    have proper-symbols ?zs
      using Suc.prems(1) by simp
    moreover have symbols-lt 5 ?zs
    proof standard+
      fix i :: nat
      assume i < length ?zs
      then have i < j
        using j by simp
      then have ?zs ! i < 6
        using Suc.prems(2) j by simp
      moreover have ?zs ! i ≠ ♯
        using ‹i < j› j by simp
      ultimately show ?zs ! i < ♯
        by simp
```

qed
moreover have $\neg$ *has2 ?zs* **0** |
using *not-has2-take Suc.prems*(*4*) **by** *simp*
moreover have *?zs* $\neq$ [] $\longrightarrow$ *last ?zs* = |
proof
assume *neq-Nil*: *?zs* $\neq$ []
then have $j > 0$
by *simp*
moreover have *xs* ! *j* = ♯
using *j* **by** *simp*
ultimately have *xs* ! *Suc* ($j - 1$) = ♯
by *simp*
moreover have $j - 1 <$ *length xs* $- 1$
by (*simp add*: *Suc-leI* ‹$0 < j$› *diff-less-mono j*(*1*))
ultimately have *xs* ! ($j - 1$) $\neq$ **0** *xs* ! ($j - 1$) $\neq$ **1**
using *Suc.prems has2-def* **by** *auto*
then have *?zs* ! ($j - 1$) $\neq$ **0** *?zs* ! ($j - 1$) $\neq$ **1**
by (*simp-all add*: ‹$0 < j$›)
moreover have *?zs* ! ($j - 1$) < ♯
using ‹*symbols-lt 5 ?zs*› ‹$0 < j$ › *j*(*1*) *len*
by *simp*
moreover have *?zs* ! ($j - 1$) $\geq$ **0**
using ‹*proper-symbols ?zs*› *len* ‹$0 < j$› **by** (*metis One-nat-def Suc-1 Suc-leI diff-less zero-less-one*)
ultimately have *?zs* ! ($j - 1$) = |
by *simp*
then show *last ?zs* = |
using *len* **by** (*metis last-conv-nth neq-Nil*)
qed
ultimately show *numlist-wf ?zs*
using *numlist-wf-iff* **by** *simp*
qed

let *?ts* = *take* (*Suc j*) *xs*
have *?ts* = *?zs* @ [♯]
using *j* **by** (*metis take-Suc-conv-app-nth*)
then have *numlistlist-wf ?ts*
using *numlistlist-wf-numlist-wf* ‹*numlist-wf ?zs*› **by** *simp*
moreover have *xs* = *?ts* @ *?ys*
by *simp*
ultimately show *numlistlist-wf xs*
using *wf-ys numlistlist-wf-append* **by** *fastforce*
qed

lemma *numlistlist-not-has2*:
assumes $i <$ *length* (*numlistlist nss*) $- 1$ **and** *numlistlist nss* ! *i* = **0**
shows *numlistlist nss* ! (*Suc i*) $\neq$ |
using *assms*
proof (*induction nss arbitrary*: *i*)
case *Nil*
then show *?case*
by (*simp add*: *numlistlist-Nil*)
next
case (*Cons ns nss*)
show *numlistlist* (*ns* # *nss*) ! (*Suc i*) $\neq$ |
proof (*cases* $i <$ *length* (*numlistlist* [*ns*]))
case *True*
have *numlistlist* (*ns* # *nss*) ! *i* = (*numlistlist* [*ns*] @ *numlistlist nss*) ! *i*
using *numlistlist-def* **by** *simp*
then have *numlistlist* (*ns* # *nss*) ! *i* = *numlistlist* [*ns*] ! *i*
using *True* **by** (*simp add*: *nth-append*)
then have *numlistlist* (*ns* # *nss*) ! *i* = (*numlist ns* @ [♯]) ! *i*
using *numlistlist-def* **by** *simp*
moreover have *numlistlist* (*ns* # *nss*) ! *i* = **0**

```
      using Cons by simp
    ultimately have (numlist ns @ [♯]) ! i = 0
      by simp
    moreover have (numlist ns @ [♯]) ! (length (numlist ns @ [♯]) − 1) = ♯
      by simp
    ultimately have i ≠ length (numlist ns @ [♯]) − 1
      by auto
    then have ∗: i ≠ length (numlistlist [ns]) − 1
      using numlistlist-def by simp
    then have ∗∗: i < length (numlistlist [ns]) − 1
      using True by simp
    then have ∗∗∗: i < length (numlist ns)
      using numlistlist-def by simp
    then have ns ≠ []
      using numlist-Nil by auto
    then have last (numlist ns) = |
      by (metis last-numlist-4 numlist-Nil numlist-inj)

    have 3: numlist ns ! j = numlistlist (ns # nss) ! j if j < length (numlist ns) for j
    proof −
      have j: j < length (numlistlist [ns])
        using that numlistlist-def by simp
      have numlistlist (ns # nss) ! j = (numlistlist [ns] @ numlistlist nss) ! j
        using numlistlist-def by simp
      then have numlistlist (ns # nss) ! j = numlistlist [ns] ! j
        using j by (simp add: nth-append)
      then have numlistlist (ns # nss) ! j = (numlist ns @ [♯]) ! j
        using numlistlist-def by simp
      then show ?thesis
        by (simp add: nth-append that)
    qed
    have 4: numlistlist (ns # nss) ! (length (numlist ns)) = ♯
      by (simp add: numlistlist-def)

    show ?thesis
    proof (cases i = length (numlist ns) − 1)
      case True
      then show ?thesis
        using 3 4 ∗∗∗ by (metis Suc-le-D Suc-le-eq diff-Suc-1 eval-nat-numeral(3) n-not-Suc-n)
    next
      case False
      then have i < length (numlist ns) − 1
        using ∗∗∗ by simp
      then show ?thesis
        using numlist-not-has2 ∗∗∗ 3 ‹ns ≠ []›
        by (metis Cons.prems(2) Suc-diff-1 length-greater-0-conv not-less-eq numlist-Nil numlist-inj)
    qed
  next
    case False
    then have i ≥ length (numlistlist [ns])
      by simp
    let ?i = i − length (numlistlist [ns])
    have numlistlist (ns # nss) ! i = (numlistlist [ns] @ numlistlist nss) ! i
      using numlistlist-def by simp
    then have numlistlist (ns # nss) ! i = numlistlist nss ! ?i
      using False by (simp add: nth-append)
    then have numlistlist nss ! ?i = 0
      using Cons by simp
    moreover have ?i < length (numlistlist nss) − 1
    proof −
      have length (numlistlist (ns # nss)) = length (numlistlist [ns]) + length (numlistlist nss)
        using numlistlist-def by simp
      then show ?thesis
```

        using *False Cons* **by** *simp*
    qed
    ultimately have *numlistlist nss* ! *Suc ?i* ≠ |
      using *Cons* **by** *simp*
    moreover have *numlistlist* (*ns* # *nss*) ! *Suc i* = *numlistlist nss* ! *Suc ?i*
      using *False numlistlist-append*
     by (*smt* (*verit*, *del-insts*) *Suc-diff-Suc Suc-lessD append-Cons append-Nil diff-Suc-Suc not-less-eq nth-append*)
    ultimately show *?thesis*
      by *simp*
  qed
qed

lemma *numlistlist-not-has2'*:
  assumes $i < length\ (numlistlist\ nss) - 1$ **and** *numlistlist nss* ! i = **0** ∨ *numlistlist nss* ! i = **1**
  shows *numlistlist nss* ! (*Suc i*) ≠ ♯
  using *assms*
proof (*induction nss arbitrary*: *i*)
  case *Nil*
  then show *?case*
    by (*simp add*: *numlistlist-Nil*)
next
  case (*Cons ns nss*)
  show *numlistlist* (*ns* # *nss*) ! (*Suc i*) ≠ ♯
  proof (*cases i* < *length* (*numlistlist* [*ns*]))
    case *True*
    have *numlistlist* (*ns* # *nss*) ! *i* = (*numlistlist* [*ns*] @ *numlistlist nss*) ! *i*
      using *numlistlist-def* **by** *simp*
    then have *numlistlist* (*ns* # *nss*) ! *i* = *numlistlist* [*ns*] ! *i*
      using *True* **by** (*simp add*: *nth-append*)
    then have *numlistlist* (*ns* # *nss*) ! *i* = (*numlist ns* @ [♯]) ! *i*
      using *numlistlist-def* **by** *simp*
    moreover have *numlistlist* (*ns* # *nss*) ! *i* = **0** ∨ *numlistlist* (*ns* # *nss*) ! *i* = **1**
      using *Cons* **by** *simp*
    ultimately have (*numlist ns* @ [♯]) ! *i* = **0** ∨ (*numlist ns* @ [♯]) ! *i* = **1**
      by *simp*
    moreover have (*numlist ns* @ [♯]) ! (*length* (*numlist ns* @ [♯]) − *1*) = ♯
      by *simp*
    ultimately have *i* ≠ *length* (*numlist ns* @ [♯]) − *1*
      by *auto*
    then have *i* ≠ *length* (*numlistlist* [*ns*]) − *1*
      using *numlistlist-def* **by** *simp*
    then have *i* < *length* (*numlistlist* [*ns*]) − *1*
      using *True* **by** *simp*
    then have ∗: *i* < *length* (*numlist ns*)
      using *numlistlist-def* **by** *simp*
    then have *ns* ≠ []
      using *numlist-Nil* **by** *auto*
    then have *last* (*numlist ns*) = |
      by (*metis last-numlist-4 numlist-Nil numlist-inj*)

    have ∗∗: *numlist ns* ! *j* = *numlistlist* (*ns* # *nss*) ! *j* **if** *j* < *length* (*numlist ns*) **for** *j*
    proof −
      have *j*: *j* < *length* (*numlistlist* [*ns*])
        using *that numlistlist-def* **by** *simp*
      have *numlistlist* (*ns* # *nss*) ! *j* = (*numlistlist* [*ns*] @ *numlistlist nss*) ! *j*
        using *numlistlist-def* **by** *simp*
      then have *numlistlist* (*ns* # *nss*) ! *j* = *numlistlist* [*ns*] ! *j*
        using *j* **by** (*simp add*: *nth-append*)
      then have *numlistlist* (*ns* # *nss*) ! *j* = (*numlist ns* @ [♯]) ! *j*
        using *numlistlist-def* **by** *simp*
      then show *?thesis*
        by (*simp add*: *nth-append that*)
    qed

show *?thesis*
proof (*cases* $i = length$ (*numlist ns*) $- 1$)
case *True*
then show *?thesis*
using ‹*last* (*numlist ns*) = |› ‹*ns* ≠ []› *Cons.prems*(*2*) ∗ ∗∗ *numlist-Nil numlist-inj*
by (*metis last-conv-nth num.simps*(*8*) *numeral-eq-iff semiring-norm*(*83*) *verit-eq-simplify*(*8*))
next
case *False*
then have $i <$ *length* (*numlist ns*) $- 1$
using ∗ **by** *simp*
then show *?thesis*
using ∗ ∗∗ *symbols-lt-numlist numlist-not-has2* **by** (*metis Suc-lessI diff-Suc-1 less-irrefl-nat*)
qed
next
case *False*
then have $i \geq$ *length* (*numlistlist* [*ns*])
by *simp*
let *?i* = *i* − *length* (*numlistlist* [*ns*])
have *numlistlist* (*ns* # *nss*) ! *i* = (*numlistlist* [*ns*] @ *numlistlist nss*) ! *i*
using *numlistlist-def* **by** *simp*
then have *numlistlist* (*ns* # *nss*) ! *i* = *numlistlist nss* ! *?i*
using *False* **by** (*simp add*: *nth-append*)
then have *numlistlist nss* ! *?i* = **0** ∨ *numlistlist nss* ! *?i* = **1**
using *Cons* **by** *simp*
moreover have *?i* < *length* (*numlistlist nss*) − *1*
proof −
have *length* (*numlistlist* (*ns* # *nss*)) = *length* (*numlistlist* [*ns*]) + *length* (*numlistlist nss*)
using *numlistlist-def* **by** *simp*
then show *?thesis*
using *False Cons* **by** *simp*
qed
ultimately have *numlistlist nss* ! *Suc ?i* ≠ ♯
using *Cons* **by** *simp*
moreover have *numlistlist* (*ns* # *nss*) ! *Suc i* = *numlistlist nss* ! *Suc ?i*
using *False numlistlist-append*
by (*smt* (*verit*, *del-insts*) *Suc-diff-Suc Suc-lessD append-Cons append-Nil diff-Suc-Suc not-less-eq nth-append*)
ultimately show *?thesis*
by *simp*
qed
qed

lemma *last-numlistlist-5*: *numlistlist nss* ≠ [] ⟹ *last* (*numlistlist nss*) = ♯
using *numlistlist-def* **by** (*induction nss*) *simp-all*

lemma *numlistlist-wf-has2′*:
assumes *numlistlist-wf xs*
shows *proper-symbols-lt 6 xs* ∧ (*xs* ≠ [] ⟶ *last xs* = ♯) ∧ ¬ *has2 xs* **0** | ∧ ¬ *has2 xs* **0** ♯ ∧ ¬ *has2 xs* **1** ♯
proof −
obtain *nss* **where** *nss*: *numlistlist nss* = *xs*
using *numlistlist-wf-def assms* **by** *auto*
have *proper-symbols xs*
using *nss proper-symbols-numlistlist* **by** *auto*
moreover have *symbols-lt 6 xs*
using *nss symbols-lt-numlistlist* **by** *auto*
moreover have *xs* ≠ [] ⟶ *last xs* = ♯
using *nss last-numlistlist-5* **by** *auto*
moreover have ¬ *has2 xs* **0** | **and** ¬ *has2 xs* **0** ♯ **and** ¬ *has2 xs* **1** ♯
using *numlistlist-not-has2 numlistlist-not-has2′ has2-def nss* **by** *auto*
ultimately show *?thesis*
by *simp*
qed

```
lemma numlistlist-wf-iff:
  numlistlist-wf xs ⟷
   proper-symbols-lt 6 xs ∧ (xs ≠ [] ⟶ last xs = ♯) ∧ ¬ has2 xs 𝟎 | ∧ ¬ has2 xs 𝟎 ♯ ∧ ¬ has2 xs 𝟏 ♯
  using numlistlist-wf-has2 numlistlist-wf-has2' by blast
```

2.12.3 A Turing machine to check for subsequences of length two

In order to implement the well-formedness criteria we need to be able to check a symbol sequence for subsequences of length two. The next Turing machine has symbol parameters y and z and checks whether the sequence $[y,\ z]$ exists on tape j_1. It writes to tape j_2 the number 0 or 1 if the sequence is present or not, respectively.

```
definition tm-not-has2 :: symbol ⇒ symbol ⇒ tapeidx ⇒ tapeidx ⇒ machine where
  tm-not-has2 y z j1 j2 ≡
    tm-set j2 [𝟎, 𝟎] ;;
    WHILE [] ; λrs. rs ! j1 ≠ □ DO
      IF λrs. rs ! j2 = 𝟏 ∧ rs ! j1 = z THEN
        tm-right j2 ;;
        tm-write j2 𝟏 ;;
        tm-left j2
      ELSE
        []
      ENDIF ;;
      tm-trans2 j1 j2 (λh. if h = y then 𝟏 else 𝟎) ;;
      tm-right j1
    DONE ;;
    tm-right j2 ;;
    IF λrs. rs ! j2 = 𝟏 THEN
      tm-set j2 (canrepr 1)
    ELSE
      tm-set j2 (canrepr 0)
    ENDIF ;;
    tm-cr j1 ;;
    tm-not j2

lemma tm-not-has2-tm:
  assumes k ≥ 2 and G ≥ 4 and 0 < j2 and j1 < k and j2 < k
  shows turing-machine k G (tm-not-has2 y z j1 j2)
proof −
  have symbols-lt G [𝟎, 𝟎]
    using assms(2) by (simp add: nth-Cons')
  moreover have symbols-lt G (canrepr 0)
    using assms(2) by simp
  moreover have symbols-lt G (canrepr 1)
    using assms(2) canrepr-1 by simp
  ultimately show ?thesis
  unfolding tm-not-has2-def
  using assms tm-right-tm tm-write-tm tm-left-tm tm-cr-tm Nil-tm tm-trans2-tm tm-set-tm
    turing-machine-loop-turing-machine turing-machine-branch-turing-machine tm-not-tm
  by simp
qed

locale turing-machine-has2 =
  fixes y z :: symbol and j1 j2 :: tapeidx
begin

context
  fixes tps0 :: tape list and xs :: symbol list and k :: nat
  assumes xs: proper-symbols xs
  assumes yz: y > 1 z > 1
  assumes jk: j1 < k j2 < k j1 ≠ j2 0 < j2 length tps0 = k
  assumes tps0:
    tps0 ! j1 = (⌊xs⌋, 1)
    tps0 ! j2 = (⌊[]⌋, 1)
```

begin

definition *tm1* ≡ *tm-set j2* [**0**, **0**]

definition *tmT1* ≡ *tm-right j2*
definition *tmT2* ≡ *tmT1* ;; *tm-write j2* **1**
definition *tmT3* ≡ *tmT2* ;; *tm-left j2*

definition *tmL1* ≡ *IF* λ*rs. rs* ! *j2* = **1** ∧ *rs* ! *j1* = *z THEN tmT3 ELSE* [] *ENDIF*
definition *tmL2* ≡ *tmL1* ;; *tm-trans2 j1 j2* (λ*h. if h* = *y then* **1** *else* **0**)
definition *tmL3* ≡ *tmL2* ;; *tm-right j1*
definition *tmL* ≡ *WHILE* [] ; λ*rs. rs* ! *j1* ≠ □ *DO tmL3 DONE*

definition *tm2* ≡ *tm1* ;; *tmL*
definition *tm3* ≡ *tm2* ;; *tm-right j2*
definition *tm34* ≡ *IF* λ*rs. rs* ! *j2* = **1** *THEN tm-set j2* (*canrepr 1*) *ELSE tm-set j2* (*canrepr 0*) *ENDIF*
definition *tm4* ≡ *tm3* ;; *tm34*
definition *tm5* ≡ *tm4* ;; *tm-cr j1*
definition *tm6* ≡ *tm5* ;; *tm-not j2*

lemma *tm6-eq-tm-not-has2*: *tm6* = *tm-not-has2 y z j1 j2*
  unfolding *tm6-def tm5-def tm4-def tm34-def tm3-def tm2-def tmL-def tmL3-def tmL2-def tmL1-def*
    tmT3-def tmT2-def tmT1-def tm1-def tm-not-has2-def
  by *simp*

definition *tps1* :: *tape list* **where**
  tps1 ≡ *tps0*
    [*j1* := (⌊*xs*⌋, *1*),
     j2 := (⌊[**0**, **0**]⌋, *1*)]

lemma *tm1*: *transforms tm1 tps0 14 tps1*
  unfolding *tm1-def*
proof (*tform tps*: *tps0 tps1-def jk*)
  show ∀ *i*<*length* [**0**, **0**]. *Suc 0* < [**0**, **0**] ! *i*
    by (*simp add*: *nth-Cons'*)
  show *tps1* = *tps0*[*j2* := (⌊[**0**, **0**]⌋, *1*)]
    using *tps1-def tps0 jk* **by** (*metis list-update-id*)
qed

abbreviation *has-at* :: *nat* ⇒ *bool* **where**
  has-at i ≡ *xs* ! *i* = *y* ∧ *xs* ! *Suc i* = *z*

definition *tpsL* :: *nat* ⇒ *tape list* **where**
  tpsL t ≡ *tps0*
    [*j1* := (⌊*xs*⌋, *Suc t*),
     j2 := (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, *if* ∃ *i*<*t* − *1*. *has-at i then* **1** *else* **0**]⌋, *1*)]

lemma *tpsL-eq-tps1*: *tpsL 0* = *tps1*
  unfolding *tps1-def tpsL-def* **using** *yz jk* **by** *simp*

lemma *tm1'* [*transforms-intros*]: *transforms tm1 tps0 14* (*tpsL 0*)
  using *tm1 tpsL-eq-tps1* **by** *simp*

definition *tpsT1* :: *nat* ⇒ *tape list* **where**
  tpsT1 t ≡ *tps0*
    [*j1* := (⌊*xs*⌋, *Suc t*),
     j2 := (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, *if* ∃ *i*<*t* − *1*. *has-at i then* **1** *else* **0**]⌋, *2*)]

definition *tpsT2* :: *nat* ⇒ *tape list* **where**
  tpsT2 t ≡ *tps0*
    [*j1* := (⌊*xs*⌋, *Suc t*),
     j2 := (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, *if* ∃ *i*<*t*. *has-at i then* **1** *else* **0**]⌋, *2*)]

definition *tpsT3* :: *nat* ⇒ *tape list* **where**
  tpsT3 t ≡ *tps0*
    [*j1* := (⌊*xs*⌋, *Suc t*),
     j2 := (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, *if* ∃*i*<*t*. *has-at i then* **1** *else* **0**]⌋, *1*)]

lemma *contents-1-update*: (⌊[*a*, *b*]⌋, *1*) |:=| *v* = (⌊[*v*, *b*]⌋, *1*) **for** *a b v* :: *symbol*
  using *contents-def* **by** *auto*

lemma *contents-2-update*: (⌊[*a*, *b*]⌋, *2*) |:=| *v* = (⌊[*a*, *v*]⌋, *2*) **for** *a b v* :: *symbol*
  using *contents-def* **by** *auto*

context
  fixes *t* :: *nat*
  assumes *then-branch*: ⌊*xs*⌋ *t* = *y xs* ! *t* = *z*
begin

lemma *tmT1* [*transforms-intros*]: *transforms tmT1* (*tpsL t*) *1* (*tpsT1 t*)
  unfolding *tmT1-def*
proof (*tform tps*: *tpsL-def tpsT1-def jk*)
  have *tpsL t* ! *j2* |+| *1* = (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, *if* ∃*i*<*t* − *1*. *has-at i then* **1** *else* **0**]⌋, *2*)
    using *jk tpsL-def* **by** *simp*
  moreover have *tpsT1 t* = (*tpsL t*)[*j2* := (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, *if* ∃*i*<*t* − *1*. *has-at i then* **1** *else* **0**]⌋,
2)]
    unfolding *tpsT1-def tpsL-def* **by** *simp*
  ultimately show *tpsT1 t* = (*tpsL t*)[*j2* := *tpsL t* ! *j2* |+| *1*]
    by *presburger*
qed

lemma *tmT2* [*transforms-intros*]: *transforms tmT2* (*tpsL t*) *2* (*tpsT2 t*)
  unfolding *tmT2-def*
proof (*tform tps*: *tpsT1-def tpsT2-def jk*)
  have *1*: *tpsT1 t* ! *j2* = (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, *if* ∃*i*<*t* − *1*. *has-at i then* **1** *else* **0**]⌋, *2*)
    using *tpsT1-def jk* **by** *simp*
  have *2*: *tpsT1 t* ! *j2* |:=| **1** = (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, **1**]⌋, *2*)
    using *tpsT1-def jk contents-2-update* **by** *simp*
  have *3*: *tpsT2 t* ! *j2* = (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, *if* ∃*i*<*t*. *has-at i then* **1** *else* **0**]⌋, *2*)
    using *tpsT2-def jk* **by** *simp*

  have ∃*i*<*t*. *has-at i*
  proof −
    have *t* > *0*
      using *then-branch*(*1*) *yz*(*1*) **by** (*metis contents-at-0 gr0I less-numeral-extra*(*4*))
    then have *y* = *xs* ! (*t* − *1*)
      using *then-branch*(*1*) **by** (*metis contents-def nat-neq-iff not-one-less-zero yz*(*1*))
    moreover have *z* = *xs* ! *t*
      using *then-branch*(*2*) **by** *simp*
    ultimately have *has-at* (*t* − *1*)
      using ‹*0* < *t*› **by** *simp*
    then show ∃*i*<*t*. *has-at i*
      using ‹*0* < *t*› **by** (*metis Suc-pred′ lessI*)
  qed
  then have (*if* ∃*i*<*t*. *has-at i then* **1** *else* **0**) = **1**
    by *simp*
  then have *tpsT1 t* ! *j2* |:=| **1** = (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, *if* ∃*i*<*t*. *has-at i then* **1** *else* **0**]⌋, *2*)
    using *2 3* **by** (*smt* (*verit*, *ccfv-threshold*))
  then show *tpsT2 t* = (*tpsT1 t*)[*j2* := *tpsT1 t* ! *j2* |:=| **1**]
    unfolding *tpsT2-def tpsT1-def* **using** *jk* **by** *simp*
qed

lemma *tmT3* [*transforms-intros*]: *transforms tmT3* (*tpsL t*) *3* (*tpsT3 t*)
  unfolding *tmT3-def* **by** (*tform tps*: *tpsT2-def tpsT3-def jk*)

end

lemma *tmL1* [*transforms-intros*]:
  assumes *ttt* = *5* **and** *t* < *length xs*
  shows *transforms tmL1* (*tpsL t*) *ttt* (*tpsT3 t*)
  unfolding *tmL1-def*
proof (*tform tps*: *assms*(*1*) *tpsL-def tpsT3-def jk*)
  have *read* (*tpsL t*) ! *j1* = *tpsL t* :.: *j1*
    using *jk tpsL-def tapes-at-read′*[*of j1 tpsL t*] **by** *simp*
  then have *1*: *read* (*tpsL t*) ! *j1* = *xs* ! *t*
    using *jk tpsL-def assms*(*2*) **by** *simp*
  then show *read* (*tpsL t*) ! *j2* = **1** ∧ *read* (*tpsL t*) ! *j1* = *z* ⟹ *xs* ! *t* = *z*
    by *simp*
  have *read* (*tpsL t*) ! *j2* = *tpsL t* :.: *j2*
    using *jk tpsL-def tapes-at-read′*[*of j2 tpsL t*] **by** *simp*
  then have *2*: *read* (*tpsL t*) ! *j2* = (*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**)
    using *jk tpsL-def* **by** *simp*
  then show *read* (*tpsL t*) ! *j2* = **1** ∧ *read* (*tpsL t*) ! *j1* = *z* ⟹ ⌊*xs*⌋ *t* = *y*
    by *presburger*
  show *tpsT3 t* = *tpsL t* **if** ¬ (*read* (*tpsL t*) ! *j2* = **1** ∧ *read* (*tpsL t*) ! *j1* = *z*)
  proof −
    have (∃ *i*<*t*. *has-at i*) = (∃ *i*<*t* − *1*. *has-at i*)
    proof (*cases t* = *0*)
      case *True*
      then show *?thesis*
        by *simp*
    next
      case *False*
      have ¬ ((*if* ⌊*xs*⌋ *t* = *y then 0*::*symbol else 1*) = *0* ∧ *xs* ! *t* = *z*)
        using *1 2 that* **by** *simp*
      then have ¬ (⌊*xs*⌋ *t* = *y* ∧ *xs* ! *t* = *z*)
        by *auto*
      then have ¬ (*has-at* (*t* − *1*))
        using *False Suc-pred′ assms*(*2*) *contents-inbounds less-imp-le-nat* **by** *simp*
      moreover have (∃ *i*<*t*. *has-at i*) = (∃ *i*<*t* − *Suc 0*. *has-at i*) ∨ *has-at* (*t* − *1*)
        using *False* **by** (*metis One-nat-def add-diff-inverse-nat less-Suc-eq less-one plus-1-eq-Suc*)
      ultimately show *?thesis*
        by *auto*
    qed
    then have *eq*: (*if* ∃ *i*<*t* − *1*. *has-at i then* **1** *else* **0**) = (*if* ∃ *i*<*t*. *has-at i then* **1** *else* **0**)
      by *simp*
    show *?thesis*
      unfolding *tpsT3-def tpsL-def* **by** (*simp only*: *eq*)
  qed
qed

definition *tpsL2* :: *nat* ⇒ *tape list* **where**
  tpsL2 t ≡ *tps0*
    [*j1* := (⌊*xs*⌋, *Suc t*),
     j2 := (⌊[*if* ⌊*xs*⌋ (*Suc t*) = *y then* **1** *else* **0**, *if* ∃ *i*<*t*. *has-at i then* **1** *else* **0**]⌋, *1*)]

lemma *tmL2* [*transforms-intros*]:
  assumes *ttt* = *6* **and** *t* < *length xs*
  shows *transforms tmL2* (*tpsL t*) *ttt* (*tpsL2 t*)
  unfolding *tmL2-def*
proof (*tform tps*: *assms tpsL-def tpsT3-def jk*)
  have *tpsT3 t* ! *j2* = (⌊[*if* ⌊*xs*⌋ *t* = *y then* **1** *else* **0**, *if* ∃ *i*<*t*. *has-at i then* **1** *else* **0**]⌋, *1*)
    using *jk tpsT3-def* **by** *simp*
  then have *tpsT3 t* ! *j2* |:=| (*if tpsT3 t* :.: *j1* = *y then* **1** *else* **0**) =
      (⌊[*if tpsT3 t* :.: *j1* = *y then* **1** *else* **0**, *if* ∃ *i*<*t*. *has-at i then* **1** *else* **0**]⌋, *1*)
    using *contents-1-update* **by** *simp*
  moreover have *tpsT3 t* :.: *j1* = ⌊*xs*⌋ (*Suc t*)
    using *assms*(*2*) *tpsT3-def jk* **by** *simp*
  ultimately have *tpsT3 t* ! *j2* |:=| (*if tpsT3 t* :.: *j1* = *y then* **1** *else* **0**) =

$(\lfloor[if\ \lfloor xs\rfloor\ (Suc\ t) = y\ then\ \mathbf{1}\ else\ \mathbf{0},\ if\ \exists i<t.\ has\text{-}at\ i\ then\ \mathbf{1}\ else\ \mathbf{0}]\rfloor,\ 1)$
by *auto*
moreover have $tpsL2\ t = (tpsT3\ t)[j2 := (\lfloor[if\ \lfloor xs\rfloor\ (Suc\ t) = y\ then\ \mathbf{1}\ else\ \mathbf{0},\ if\ \exists i<t.\ has\text{-}at\ i\ then\ \mathbf{1}\ else\ \mathbf{0}]\rfloor,\ 1)]$
using *tpsL2-def tpsT3-def* **by** *simp*
ultimately show $tpsL2\ t = (tpsT3\ t)[j2 := tpsT3\ t\ !\ j2\ |{:=}|\ (if\ tpsT3\ t\ {:.:}\ j1 = y\ then\ \mathbf{1}\ else\ \mathbf{0})]$
by *presburger*
qed

lemma *tmL3* [*transforms-intros*]:
assumes $ttt = 7$ **and** $t < length\ xs$
shows *transforms tmL3* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
unfolding *tmL3-def*
proof (*tform tps*: *assms tpsL-def tpsL2-def jk*)
have $tpsL2\ t\ !\ j1 = (\lfloor xs\rfloor,\ Suc\ t)$
using *tpsL2-def jk* **by** *simp*
then show $tpsL\ (Suc\ t) = (tpsL2\ t)[j1 := tpsL2\ t\ !\ j1\ |{+}|\ 1]$
using *tpsL2-def tpsL-def jk* **by** (*simp add*: *list-update-swap*)
qed

lemma *tmL* [*transforms-intros*]:
assumes $ttt = 9 * length\ xs + 1$
shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsL* (*length xs*))
unfolding *tmL-def*
proof (*tform time*: *assms*)
have $read\ (tpsL\ t)\ !\ j1 = tpsL\ t\ {:.:}\ j1$ **for** *t*
using *tpsL-def tapes-at-read′ jk*
by (*metis* (*no-types*, *lifting*) *length-list-update*)
then have $read\ (tpsL\ t)\ !\ j1 = \lfloor xs\rfloor\ (Suc\ t)$ **for** *t*
using *tpsL-def jk* **by** *simp*
then show $\bigwedge t.\ t < length\ xs \Longrightarrow read\ (tpsL\ t)\ !\ j1 \neq \Box$ **and** $\neg\ read\ (tpsL\ (length\ xs))\ !\ j1 \neq \Box$
using *xs* **by** *auto*
qed

lemma *tm2* [*transforms-intros*]:
assumes $ttt = 9 * length\ xs + 15$
shows *transforms tm2 tps0 ttt* (*tpsL* (*length xs*))
unfolding *tm2-def* **by** (*tform tps*: *assms tpsL-def jk*)

definition *tps3* :: *tape list* **where**
$tps3 \equiv tps0$
$[j1 := (\lfloor xs\rfloor,\ Suc\ (length\ xs)),$
$j2 := (\lfloor[if\ \lfloor xs\rfloor\ (length\ xs) = y\ then\ \mathbf{1}\ else\ \mathbf{0},\ if\ \exists i<length\ xs - 1.\ has\text{-}at\ i\ then\ \mathbf{1}\ else\ \mathbf{0}]\rfloor,\ 2)]$

lemma *tm3* [*transforms-intros*]:
assumes $ttt = 9 * length\ xs + 16$
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def*
proof (*tform tps*: *assms tpsL-def jk*)
show $tps3 = (tpsL\ (length\ xs))[j2 := tpsL\ (length\ xs)\ !\ j2\ |{+}|\ 1]$
unfolding *tps3-def tpsL-def*
using *jk*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 add.right-neutral add-Suc-right fst-conv length-list-update list-update-overwrite nth-list-update-eq snd-conv*)
qed

definition *tps4* :: *tape list* **where**
$tps4 \equiv tps0$
$[j1 := (\lfloor xs\rfloor,\ Suc\ (length\ xs)),$
$j2 := (\lfloor\exists i<length\ xs - Suc\ 0.\ has\text{-}at\ i\rfloor_B,\ 1)]$

lemma *tm34* [*transforms-intros*]:
assumes $ttt = 19$

shows *transforms tm34 tps3 ttt tps4*
unfolding *tm34-def*
proof (*tform tps*: *assms tps4-def tps3-def jk*)
let *?pair* = [*if* ⌊*xs*⌋ (*length xs*) = *y then* **1** *else* **0**, *if* ∃ *i*<*length xs* − *Suc 0*. *has-at i then* **1** *else* **0**]
show *1*: *proper-symbols ?pair* **and** *proper-symbols ?pair*
by (*simp-all add*: *nth-Cons′*)
show *proper-symbols* (*canrepr 1*)
using *proper-symbols-canrepr* **by** *simp*

have *read tps3* ! *j2* = (*if* ∃ *i*<*length xs* − *Suc 0*. *has-at i then* **1** *else* **0**)
using *jk tps3-def tapes-at-read′*[*of j2 tps3*] **by** *simp*
then have ∗: *read tps3* ! *j2* = **1** ⟷ (∃ *i*<*length xs* − *Suc 0*. *has-at i*)
by *simp*

show *clean-tape* (*tps3* ! *j2*) *clean-tape* (*tps3* ! *j2*)
using *jk tps3-def clean-contents-proper*[*OF 1*] **by** *simp-all*
show *tps4* = *tps3*[*j2* := (⌊*1*⌋$_N$, *1*)] **if** *read tps3* ! *j2* = **1**
proof −
have ∃ *i*<*length xs* − *Suc 0*. *has-at i*
using *that* ∗ **by** *simp*
then have (⌊∃ *i*<*length xs* − *Suc 0*. *has-at i*⌋$_B$, *1*) = (⌊*1*⌋$_N$, *1*)
by *simp*
then have *tps4* = *tps0*
[*j1* := (⌊*xs*⌋, *Suc* (*length xs*)),
j2 := (⌊*1*⌋$_N$, *1*)]
using *tps4-def* **by** *simp*
then show *?thesis*
using *tps3-def* **by** *simp*
qed
show *tps4* = *tps3*[*j2* := (⌊*0*⌋$_N$, *1*)] **if** *read tps3* ! *j2* ≠ **1**
proof −
have ¬ (∃ *i*<*length xs* − *Suc 0*. *has-at i*)
using *that* ∗ **by** *simp*
then have (⌊∃ *i*<*length xs* − *Suc 0*. *has-at i*⌋$_B$, *1*) = (⌊*0*⌋$_N$, *1*)
by *auto*
then have *tps4* = *tps0*
[*j1* := (⌊*xs*⌋, *Suc* (*length xs*)),
j2 := (⌊*0*⌋$_N$, *1*)]
using *tps4-def* **by** *simp*
then show *?thesis*
using *tps3-def* **by** *simp*
qed

have *tps3* :#: *j2* = *2*
using *jk tps3-def* **by** *simp*
then show *8* + *tps3* :#: *j2* + *2* ∗ *length ?pair* + *Suc* (*2* ∗ *nlength 1*) + *2* ≤ *ttt*
and *8* + *tps3* :#: *j2* + *2* ∗ *length ?pair* + *Suc* (*2* ∗ *nlength 0*) + *1* ≤ *ttt*
using *assms nlength-1-simp* **by** *simp-all*
qed

lemma *tm4*:
assumes *ttt* = *9* ∗ *length xs* + *35*
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def* **by** (*tform tps*: *assms*)

definition *tps4′* :: *tape list* **where**
tps4′ ≡ *tps0*
[*j1* := (⌊*xs*⌋, *Suc* (*length xs*)),
j2 := (⌊*has2 xs y z*⌋$_B$, *1*)]

lemma *tps4′*: *tps4* = *tps4′*
using *has2-def tps4-def tps4′-def* **by** *simp*

```
lemma tm4' [transforms-intros]:
  assumes ttt = 9 * length xs + 35
  shows transforms tm4 tps0 ttt tps4'
  using assms tm4 tps4' by simp

definition tps5 :: tape list where
  tps5 ≡ tps0
    [j1 := (⌊xs⌋, 1),
     j2 := (⌊has2 xs y z ⌋B, 1)]

lemma tm5:
  assumes ttt = 10 * length xs + 38
  shows transforms tm5 tps0 ttt tps5
  unfolding tm5-def
proof (tform tps: assms tps4'-def jk)
  show clean-tape (tps4' ! j1)
    using tps4'-def jk xs by simp
  have tps4' ! j1 |#=| 1 = (⌊xs⌋, 1)
    using tps4'-def jk by simp
  then show tps5 = tps4'[j1 := tps4' ! j1 |#=| 1]
    using tps5-def tps4'-def jk by (simp add: list-update-swap)
qed

definition tps5' :: tape list where
  tps5' ≡ tps0
    [j2 := (⌊has2 xs y z⌋B, 1)]

lemma tm5' [transforms-intros]:
  assumes ttt = 10 * length xs + 38
  shows transforms tm5 tps0 ttt tps5'
proof −
  have tps5 = tps5'
    using tps5-def tps5'-def jk tps0(1) by (metis list-update-id)
  then show ?thesis
    using assms tm5 by simp
qed

definition tps6 :: tape list where
  tps6 ≡ tps0
    [j2 := (⌊¬ has2 xs y z⌋B, 1)]

lemma tm6:
  assumes ttt = 10 * length xs + 41
  shows transforms tm6 tps0 ttt tps6
  unfolding tm6-def
proof (tform tps: assms tps5'-def jk)
  have tps5'[j2 := (⌊(if has2 xs y z then 1::nat else 0) ≠ 1⌋B, 1)] =
      tps5'[j2 := (⌊¬ has2 xs y z⌋B, 1)]
    by simp
  also have ... = tps0[j2 := (⌊¬ has2 xs y z⌋B, 1)]
    using tps5'-def by simp
  also have ... = tps6
    using tps6-def by simp
  finally show tps6 = tps5'
      [j2 := (⌊(if has2 xs y z then 1::nat else 0) ≠ 1⌋B, 1)]
    by simp
qed

end

end

lemma transforms-tm-not-has2I [transforms-intros]:
```

fixes *y z* :: *symbol* **and** *j1 j2* :: *tapeidx*
fixes *tps tps'* :: *tape list* **and** *xs* :: *symbol list* **and** *ttt k* :: *nat*
assumes $j1 < k$ $j2 < k$ $j1 \neq j2$ $0 < j2$ $length\ tps = k$ $y > 1$ $z > 1$
and *proper-symbols xs*
assumes
$tps\ !\ j1 = (\lfloor xs \rfloor,\ 1)$
$tps\ !\ j2 = (\lfloor [] \rfloor,\ 1)$
assumes $ttt = 10 * length\ xs + 41$
assumes $tps' = tps$
$[j2 := (\lfloor \neg\ has2\ xs\ y\ z \rfloor_B,\ 1)]$
shows *transforms* (*tm-not-has2 y z j1 j2*) *tps ttt tps'*
proof −
interpret *loc*: *turing-machine-has2 y z j1 j2* **.**
show *?thesis*
using *loc.tps6-def loc.tm6 loc.tm6-eq-tm-not-has2 assms* **by** *metis*
qed

2.12.4 Checking well-formedness for lists

The next Turing machine checks all conditions from the criterion in lemma *numlist-wf-iff* one after another for the symbols on tape j_1 and writes to tape j_2 either the number 1 or 0 depending on whether all conditions were met. It assumes tape j_2 is initialized with 0.

definition *tm-numlist-wf* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-numlist-wf j1 j2 ≡
tm-proper-symbols-lt j1 j2 5 ;;
tm-not-has2 **0** | *j1* (*j2* + *1*) ;;
tm-and j2 (*j2* + *1*) ;;
tm-setn (*j2* + *1*) *0* ;;
tm-empty-or-endswith j1 (*j2* + *1*) | ;;
tm-and j2 (*j2* + *1*) ;;
tm-setn (*j2* + *1*) *0*

lemma *tm-numlist-wf-tm*:
assumes $k \geq 2$ **and** $G \geq 5$ **and** $0 < j2$ $0 < j1$ **and** $j1 < k$ $j2 + 1 < k$
shows *turing-machine k G* (*tm-numlist-wf j1 j2*)
using *tm-numlist-wf-def tm-proper-symbols-lt-tm tm-not-has2-tm tm-and-tm tm-setn-tm tm-empty-or-endswith-tm assms*
by *simp*

locale *turing-machine-numlist-wf* =
fixes *j1 j2* :: *tapeidx*
begin

definition *tm1* ≡ *tm-proper-symbols-lt j1 j2 5*
definition *tm2* ≡ *tm1* ;; *tm-not-has2* **0** | *j1* (*j2* + *1*)
definition *tm3* ≡ *tm2* ;; *tm-and j2* (*j2* + *1*)
definition *tm4* ≡ *tm3* ;; *tm-setn* (*j2* + *1*) *0*
definition *tm5* ≡ *tm4* ;; *tm-empty-or-endswith j1* (*j2* + *1*) |
definition *tm6* ≡ *tm5* ;; *tm-and j2* (*j2* + *1*)
definition *tm7* ≡ *tm6* ;; *tm-setn* (*j2* + *1*) *0*

lemma *tm7-eq-tm-numlist-wf*: *tm7* = *tm-numlist-wf j1 j2*
unfolding *tm7-def tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def tm-numlist-wf-def*
by *simp*

context
fixes *tps0* :: *tape list* **and** *zs* :: *symbol list* **and** *k* :: *nat*
assumes *zs*: *proper-symbols zs*
assumes *jk*: $0 < j1$ $j1 < k$ $j2 + 1 < k$ $j1 \neq j2$ $0 < j2$ $j1 \neq j2 + 1$ $length\ tps0 = k$
assumes *tps0*:
$tps0\ !\ j1 = (\lfloor zs \rfloor,\ 1)$
$tps0\ !\ j2 = (\lfloor [] \rfloor,\ 1)$
$tps0\ !\ (j2 + 1) = (\lfloor [] \rfloor,\ 1)$

begin

definition *tps1* ≡ *tps0*
 [*j2* := (⌊*proper-symbols-lt 5 zs*⌋$_B$, *1*)]

lemma *tm1* [*transforms-intros*]:
 assumes *ttt* = *5* + *7* ∗ *length zs*
 shows *transforms tm1 tps0 ttt tps1*
 unfolding *tm1-def*
 by (*tform tps*: *zs tps0 assms tps1-def jk*)

definition *tps2* ≡ *tps0*
 [*j2* := (⌊*proper-symbols-lt 5 zs*⌋$_B$, *1*),
 j2 + *1* := (⌊*if has2 zs* **0** | *then 0 else 1*⌋$_N$, *1*)]

lemma *tm2* [*transforms-intros*]:
 assumes *ttt* = *46* + *17* ∗ *length zs*
 shows *transforms tm2 tps0 ttt tps2*
 unfolding *tm2-def*
 by (*tform tps*: *zs tps0 assms tps1-def tps2-def jk*)

definition *tps3* ≡ *tps0*
 [*j2* := (⌊*proper-symbols-lt 5 zs* ∧ ¬ *has2 zs* **0** |⌋$_B$, *1*),
 j2 + *1* := (⌊*if has2 zs* **0** | *then 0 else 1*⌋$_N$, *1*)]

lemma *tm3* [*transforms-intros*]:
 assumes *ttt* = *46* + *17* ∗ *length zs* + *3*
 shows *transforms tm3 tps0 ttt tps3*
 unfolding *tm3-def* **by** (*tform tps*: *assms tps2-def tps3-def jk*)

definition *tps4* ≡ *tps0*
 [*j2* := (⌊*proper-symbols-lt 5 zs* ∧ ¬ *has2 zs* **0** |⌋$_B$, *1*),
 j2 + *1* := (⌊*0*⌋$_N$, *1*)]

lemma *tm4*:
 assumes *ttt* = *46* + *17* ∗ *length zs* + *13* + *2* ∗ *nlength* (*if has2 zs* **0** | *then 0 else 1*)
 shows *transforms tm4 tps0 ttt tps4*
 unfolding *tm4-def* **by** (*tform tps*: *assms tps3-def tps4-def jk*)

lemma *tm4′* [*transforms-intros*]:
 assumes *ttt* = *46* + *17* ∗ *length zs* + *15*
 shows *transforms tm4 tps0 ttt tps4*
 using *assms nlength-0 nlength-1-simp tm4 transforms-monotone* **by** *simp*

definition *tps5* ≡ *tps0*
 [*j2* := (⌊*proper-symbols-lt 5 zs* ∧ ¬ *has2 zs* **0** |⌋$_B$, *1*),
 j2 + *1* := (⌊*zs* = [] ∨ *last zs* = |⌋$_B$, *1*)]

lemma *tm5* [*transforms-intros*]:
 assumes *ttt* = *79* + *19* ∗ *length zs*
 shows *transforms tm5 tps0 ttt tps5*
 unfolding *tm5-def* **by** (*tform tps*: *tps4-def tps5-def jk zs tps0 assms*)

definition *tps6* ≡ *tps0*
 [*j2* := (⌊*proper-symbols-lt 5 zs* ∧ ¬ *has2 zs* **0** | ∧ (*zs* = [] ∨ *last zs* = |)⌋$_B$, *1*),
 j2 + *1* := (⌊*zs* = [] ∨ *last zs* = |⌋$_B$, *1*)]

lemma *tm6* [*transforms-intros*]:
 assumes *ttt* = *82* + *19* ∗ *length zs*
 shows *transforms tm6 tps0 ttt tps6*
 unfolding *tm6-def* **by** (*tform tps*: *tps5-def tps6-def jk time*: *assms*)

definition *tps7* ≡ *tps0*

[*j2* := (⌊*proper-symbols-lt 5 zs* ∧ ¬ *has2 zs* **0** | ∧ (*zs* = [] ∨ *last zs* = |)⌋$_B$, *1*),
j2 + *1* := (⌊*0*⌋$_N$, *1*)]

lemma *tm7*:
assumes *ttt* = *92* + *19* ∗ *length zs* + *2* ∗ *nlength* (*if zs* = [] ∨ *last zs* = | *then 1 else 0*)
shows *transforms tm7 tps0 ttt tps7*
unfolding *tm7-def* **by** (*tform tps*: *assms tps6-def tps7-def jk*)

definition *tps7′* ≡ *tps0*
[*j2* := (⌊*numlist-wf zs*⌋$_B$, *1*),
j2 + *1* := (⌊*0*⌋$_N$, *1*)]

lemma *tm7′*:
assumes *ttt* = *94* + *19* ∗ *length zs*
shows *transforms tm7 tps0 ttt tps7′*
proof −
have *ttt* ≥ *92* + *19* ∗ *length zs* + *2* ∗ *nlength* (*if zs* = [] ∨ *last zs* = | *then 1 else 0*)
using *assms nlength-1-simp* **by** *auto*
moreover have *tps7′* = *tps7*
using *tps7′-def tps7-def numlist-wf-iff* **by** *auto*
ultimately show *?thesis*
using *transforms-monotone tm7* **by** *simp*
qed

definition *tps7′′* ≡ *tps0*
[*j2* := (⌊*numlist-wf zs*⌋$_B$, *1*)]

lemma *tm7′′* [*transforms-intros*]:
assumes *ttt* = *94* + *19* ∗ *length zs*
shows *transforms tm7 tps0 ttt tps7′′*
proof −
have *tps7′′* = *tps7′*
unfolding *tps7′′-def tps7′-def* **using** *tps0 jk canrepr-0*
by (*metis add-gr-0 less-add-same-cancel1 list-update-id list-update-swap-less zero-less-two*)
then show *?thesis*
using *tm7′ assms* **by** *simp*
qed

end

end

lemma *transforms-tm-numlist-wfI* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx*
fixes *tps tps′* :: *tape list* **and** *zs* :: *symbol list* **and** *ttt k* :: *nat*
assumes *0* < *j1 j1* < *k j2* + *1* < *k j1* ≠ *j2 0* < *j2 j1* ≠ *j2* + *1 length tps* = *k*
and *proper-symbols zs*
assumes
tps ! *j1* = (⌊*zs*⌋, *1*)
tps ! *j2* = (⌊[]⌋, *1*)
tps ! (*j2* + *1*) = (⌊[]⌋, *1*)
assumes *ttt* = *94* + *19* ∗ *length zs*
assumes *tps′* = *tps*
[*j2* := (⌊*numlist-wf zs*⌋$_B$, *1*)]
shows *transforms* (*tm-numlist-wf j1 j2*) *tps ttt tps′*
proof −
interpret *loc*: *turing-machine-numlist-wf j1 j2* **.**
show *?thesis*
using *assms loc.tps7′′-def loc.tm7′′ loc.tm7-eq-tm-numlist-wf* **by** *simp*
qed

2.12.5 Checking well-formedness for lists of lists

The next Turing machine checks all conditions from the criterion in lemma *numlistlist-wf-iff* one after another for the symbols on tape j_1 and writes to tape j_2 either the number 1 or 0 depending on whether all conditions were met. It assumes tape j_2 is initialized with 0.

definition *tm-numlistlist-wf* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-numlistlist-wf j1 j2 ≡
    tm-proper-symbols-lt j1 j2 6 ;;
    tm-not-has2 **0** | *j1* (*j2* + *1*) ;;
    tm-and j2 (*j2* + *1*) ;;
    tm-setn (*j2* + *1*) *0* ;;
    tm-empty-or-endswith j1 (*j2* + *1*) ♯ ;;
    tm-and j2 (*j2* + *1*) ;;
    tm-setn (*j2* + *1*) *0* ;;
    tm-not-has2 **0** ♯ *j1* (*j2* + *1*) ;;
    tm-and j2 (*j2* + *1*) ;;
    tm-setn (*j2* + *1*) *0* ;;
    tm-not-has2 **1** ♯ *j1* (*j2* + *1*) ;;
    tm-and j2 (*j2* + *1*) ;;
    tm-setn (*j2* + *1*) *0*

lemma *tm-numlistlist-wf-tm*:
  assumes *k* ≥ *2* **and** *G* ≥ *6* **and** *0* < *j2 0* < *j1* **and** *j1* < *k j2* + *1* < *k*
  shows *turing-machine k G* (*tm-numlistlist-wf j1 j2*)
  using *tm-numlistlist-wf-def tm-proper-symbols-lt-tm tm-not-has2-tm tm-and-tm tm-setn-tm tm-empty-or-endswith-tm assms*
  by *simp*

locale *turing-machine-numlistlist-wf* =
  fixes *j1 j2* :: *tapeidx*
begin

definition *tm1* ≡ *tm-proper-symbols-lt j1 j2 6*
definition *tm2* ≡ *tm1* ;; *tm-not-has2* **0** | *j1* (*j2* + *1*)
definition *tm3* ≡ *tm2* ;; *tm-and j2* (*j2* + *1*)
definition *tm4* ≡ *tm3* ;; *tm-setn* (*j2* + *1*) *0*
definition *tm5* ≡ *tm4* ;; *tm-empty-or-endswith j1* (*j2* + *1*) ♯
definition *tm6* ≡ *tm5* ;; *tm-and j2* (*j2* + *1*)
definition *tm7* ≡ *tm6* ;; *tm-setn* (*j2* + *1*) *0*
definition *tm8* ≡ *tm7* ;; *tm-not-has2* **0** ♯ *j1* (*j2* + *1*)
definition *tm9* ≡ *tm8* ;; *tm-and j2* (*j2* + *1*)
definition *tm10* ≡ *tm9* ;; *tm-setn* (*j2* + *1*) *0*
definition *tm11* ≡ *tm10* ;; *tm-not-has2* **1** ♯ *j1* (*j2* + *1*)
definition *tm12* ≡ *tm11* ;; *tm-and j2* (*j2* + *1*)
definition *tm13* ≡ *tm12* ;; *tm-setn* (*j2* + *1*) *0*

lemma *tm13-eq-tm-numlistlist-wf*: *tm13* = *tm-numlistlist-wf j1 j2*
  unfolding *tm13-def tm12-def tm11-def tm10-def tm9-def tm8-def tm7-def tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def tm-numlistlist-wf-def*
  by *simp*

context
  fixes *tps0* :: *tape list* **and** *zs* :: *symbol list* **and** *k* :: *nat*
  assumes *zs*: *proper-symbols zs*
  assumes *jk*: *0* < *j1 j1* < *k j2* + *1* < *k j1* ≠ *j2 0* < *j2 j1* ≠ *j2* + *1 length tps0* = *k*
  assumes *tps0*:
    tps0 ! *j1* = (⌊*zs*⌋, *1*)
    tps0 ! *j2* = (⌊[]⌋, *1*)
    tps0 ! (*j2* + *1*) = (⌊[]⌋, *1*)
begin

definition *tps1* ≡ *tps0*
  [*j2* := (⌊*proper-symbols-lt 6 zs*⌋$_B$, *1*)]

lemma *tm1* [*transforms-intros*]:
  assumes $ttt = 5 + 7 * length\ zs$
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def* **by** (*tform tps*: *tps0 tps1-def zs jk time*: *assms*)

definition $tps2 \equiv tps0$
  $[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs\rfloor_B,\ 1),$
   $j2 + 1 := (\lfloor if\ has2\ zs\ \mathbf{0}\ |\ then\ 0\ else\ 1\rfloor_N,\ 1)]$

lemma *tm2* [*transforms-intros*]:
  assumes $ttt = 46 + 17 * length\ zs$
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def* **by** (*tform tps*: *zs tps0 assms tps1-def tps2-def jk*)

definition $tps3 \equiv tps0$
  $[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0}\ |\rfloor_B,\ 1),$
   $j2 + 1 := (\lfloor if\ has2\ zs\ \mathbf{0}\ |\ then\ 0\ else\ 1\rfloor_N,\ 1)]$

lemma *tm3* [*transforms-intros*]:
  assumes $ttt = 46 + 17 * length\ zs + 3$
  shows *transforms tm3 tps0 ttt tps3*
  unfolding *tm3-def* **by** (*tform tps*: *tps2-def tps3-def jk time*: *assms*)

definition $tps4 \equiv tps0$
  $[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0}\ |\rfloor_B,\ 1),$
   $j2 + 1 := (\lfloor 0\rfloor_N,\ 1)]$

lemma *tm4*:
  assumes $ttt = 46 + 17 * length\ zs + 13 + 2 * nlength\ (if\ has2\ zs\ \mathbf{0}\ |\ then\ 0\ else\ 1)$
  shows *transforms tm4 tps0 ttt tps4*
  unfolding *tm4-def* **by** (*tform tps*: *tps3-def assms tps4-def jk*)

lemma *tm4'* [*transforms-intros*]:
  assumes $ttt = 46 + 17 * length\ zs + 15$
  shows *transforms tm4 tps0 ttt tps4*
  using *assms nlength-0 nlength-1-simp tm4 transforms-monotone* **by** *simp*

definition $tps5 \equiv tps0$
  $[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0}\ |\rfloor_B,\ 1),$
   $j2 + 1 := (\lfloor zs = [] \vee last\ zs = \sharp\rfloor_B,\ 1)]$

lemma *tm5* [*transforms-intros*]:
  assumes $ttt = 79 + 19 * length\ zs$
  shows *transforms tm5 tps0 ttt tps5*
  unfolding *tm5-def* **by** (*tform tps*: *tps0 tps4-def tps5-def jk zs time*: *assms*)

definition $tps6 \equiv tps0$
  $[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0}\ | \wedge (zs = [] \vee last\ zs = \sharp)\rfloor_B,\ 1),$
   $j2 + 1 := (\lfloor zs = [] \vee last\ zs = \sharp\rfloor_B,\ 1)]$

lemma *tm6* [*transforms-intros*]:
  assumes $ttt = 82 + 19 * length\ zs$
  shows *transforms tm6 tps0 ttt tps6*
  unfolding *tm6-def* **by** (*tform tps*: *tps5-def tps6-def jk time*: *assms*)

definition $tps7 \equiv tps0$
  $[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0}\ | \wedge (zs = [] \vee last\ zs = \sharp)\rfloor_B,\ 1),$
   $j2 + 1 := (\lfloor 0\rfloor_N,\ 1)]$

lemma *tm7*:
  assumes $ttt = 92 + 19 * length\ zs + 2 * nlength\ (if\ zs = [] \vee last\ zs = \sharp\ then\ 1\ else\ 0)$
  shows *transforms tm7 tps0 ttt tps7*

unfolding *tm7-def* **by** (*tform tps*: *assms tps6-def tps7-def jk*)

lemma *tm7'* [*transforms-intros*]:
assumes $ttt = 94 + 19 * length\ zs$
shows *transforms tm7 tps0 ttt tps7*
using *transforms-monotone tm7 nlength-1-simp assms* **by** *simp*

definition $tps8 \equiv tps0$
$[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0} \mid \wedge (zs = [] \vee last\ zs = \sharp)\rfloor_B,\ 1),$
$j2 + 1 := (\lfloor if\ has2\ zs\ \mathbf{0}\ \sharp\ then\ 0\ else\ 1\rfloor_N,\ 1)]$

lemma *tm8* [*transforms-intros*]:
assumes $ttt = 135 + 29 * length\ zs$
shows *transforms tm8 tps0 ttt tps8*
unfolding *tm8-def* **by** (*tform tps*: *canrepr-0 zs tps0 assms tps7-def tps8-def jk*)

definition $tps9 \equiv tps0$
$[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0} \mid \wedge (zs = [] \vee last\ zs = \sharp) \wedge \neg\ has2\ zs\ \mathbf{0}\ \sharp\rfloor_B,\ 1),$
$j2 + 1 := (\lfloor if\ has2\ zs\ \mathbf{0}\ \sharp\ then\ 0\ else\ 1\rfloor_N,\ 1)]$

lemma *tm9* [*transforms-intros*]:
assumes $ttt = 138 + 29 * length\ zs$
shows *transforms tm9 tps0 ttt tps9*
unfolding *tm9-def* **by** (*tform tps*: *tps8-def tps9-def jk time*: *assms*)

definition $tps10 \equiv tps0$
$[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0} \mid \wedge (zs = [] \vee last\ zs = \sharp) \wedge \neg\ has2\ zs\ \mathbf{0}\ \sharp\rfloor_B,\ 1),$
$j2 + 1 := (\lfloor 0\rfloor_N,\ 1)]$

lemma *tm10*:
assumes $ttt = 148 + 29 * length\ zs + 2 * nlength\ (if\ has2\ zs\ \mathbf{0}\ \sharp\ then\ 0\ else\ 1)$
shows *transforms tm10 tps0 ttt tps10*
unfolding *tm10-def* **by** (*tform tps*: *assms tps9-def tps10-def jk*)

lemma *tm10'* [*transforms-intros*]:
assumes $ttt = 150 + 29 * length\ zs$
shows *transforms tm10 tps0 ttt tps10*
using *transforms-monotone tm10 nlength-1-simp assms* **by** *simp*

definition $tps11 \equiv tps0$
$[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0} \mid \wedge (zs = [] \vee last\ zs = \sharp) \wedge \neg\ has2\ zs\ \mathbf{0}\ \sharp\rfloor_B,\ 1),$
$j2 + 1 := (\lfloor if\ has2\ zs\ \mathbf{1}\ \sharp\ then\ 0\ else\ 1\rfloor_N,\ 1)]$

lemma *tm11* [*transforms-intros*]:
assumes $ttt = 191 + 39 * length\ zs$
shows *transforms tm11 tps0 ttt tps11*
unfolding *tm11-def* **by** (*tform tps*: *canrepr-0 zs tps0 assms tps10-def tps11-def jk*)

definition $tps12 \equiv tps0$
$[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0} \mid \wedge (zs = [] \vee last\ zs = \sharp) \wedge \neg\ has2\ zs\ \mathbf{0}\ \sharp \wedge \neg\ has2\ zs\ \mathbf{1}\ \sharp\rfloor_B,\ 1),$
$j2 + 1 := (\lfloor if\ has2\ zs\ \mathbf{1}\ \sharp\ then\ 0\ else\ 1\rfloor_N,\ 1)]$

lemma *tm12* [*transforms-intros*]:
assumes $ttt = 194 + 39 * length\ zs$
shows *transforms tm12 tps0 ttt tps12*
unfolding *tm12-def* **by** (*tform tps*: *assms tps11-def tps12-def jk*)

definition $tps13 \equiv tps0$
$[j2 := (\lfloor proper\text{-}symbols\text{-}lt\ 6\ zs \wedge \neg\ has2\ zs\ \mathbf{0} \mid \wedge (zs = [] \vee last\ zs = \sharp) \wedge \neg\ has2\ zs\ \mathbf{0}\ \sharp \wedge \neg\ has2\ zs\ \mathbf{1}\ \sharp\rfloor_B,\ 1),$
$j2 + 1 := (\lfloor 0\rfloor_N,\ 1)]$

lemma *tm13*:
assumes $ttt = 204 + 39 * length\ zs + 2 * nlength\ (if\ has2\ zs\ \mathbf{1}\ \sharp\ then\ 0\ else\ 1)$

shows *transforms tm13 tps0 ttt tps13*
unfolding *tm13-def* **by** (*tform tps*: *assms tps12-def tps13-def jk*)

lemma *tm13'*:
assumes *ttt* = *206* + *39* ∗ *length zs*
shows *transforms tm13 tps0 ttt tps13*
using *transforms-monotone tm13 nlength-1-simp assms* **by** *simp*

definition *tps13'* ≡ *tps0*
[*j2* := (⌊*proper-symbols-lt 6 zs* ∧ ¬ *has2 zs* **0** | ∧ (*zs* = [] ∨ *last zs* = ♯) ∧ ¬ *has2 zs* **0** ♯ ∧ ¬ *has2 zs* **1** ♯⌋$_B$, *1*)]

lemma *tm13''*:
assumes *ttt* = *206* + *39* ∗ *length zs*
shows *transforms tm13 tps0 ttt tps13'*
proof −
have *tps13'* = *tps13*
unfolding *tps13'-def tps13-def*
using *tps0*(*3*) *jk canrepr-0 list-update-id*[*of tps0 Suc j2*]
by (*simp add*: *list-update-swap*)
then show *?thesis*
using *tm13' assms* **by** *simp*
qed

definition *tps13''* ≡ *tps0*
[*j2* := (⌊*numlistlist-wf zs*⌋$_B$, *1*)]

lemma *tm13'''*:
assumes *ttt* = *206* + *39* ∗ *length zs*
shows *transforms tm13 tps0 ttt tps13''*
proof −
have *tps13''* = *tps13'*
using *numlistlist-wf-iff tps13''-def tps13'-def* **by** *auto*
then show *?thesis*
using *assms tm13'' numlistlist-wf-iff* **by** *simp*
qed

end

end

lemma *transforms-tm-numlistlist-wfI* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx*
fixes *tps tps'* :: *tape list* **and** *zs* :: *symbol list* **and** *ttt k* :: *nat*
assumes *0* < *j1 j1* < *k j2* + *1* < *k j1* ≠ *j2 0* < *j2 j1* ≠ *j2* + *1 length tps* = *k*
and *proper-symbols zs*
assumes
tps ! *j1* = (⌊*zs*⌋, *1*)
tps ! *j2* = (⌊[]⌋, *1*)
tps ! (*j2* + *1*) = (⌊[]⌋, *1*)
assumes *ttt* = *206* + *39* ∗ *length zs*
assumes *tps'* = *tps*
[*j2* := (⌊*numlistlist-wf zs*⌋$_B$, *1*)]
shows *transforms* (*tm-numlistlist-wf j1 j2*) *tps ttt tps'*
proof −
interpret *loc*: *turing-machine-numlistlist-wf j1 j2* **.**
show *?thesis*
using *assms loc.tps13''-def loc.tm13''' loc.tm13-eq-tm-numlistlist-wf* **by** *simp*
qed

end

Chapter 3

Time complexity

theory *NP*
  imports *Elementary Composing Symbol-Ops*
begin

In order to formulate the Cook-Levin theorem we need to formalize `SAT` and $\mathcal{NP}$-completeness. This chapter is devoted to the latter and hence introduces the complexity class $\mathcal{NP}$ and the concept of polynomial-time many-one reduction. Moreover, although not required for the Cook-Levin theorem, it introduces the class $\mathcal{P}$ and contains a proof of $\mathcal{P} \subseteq \mathcal{NP}$ (see Section 3.3). The chapter concludes with some easy results about $\mathcal{P}$, $\mathcal{NP}$ and reducibility in Section 3.4.

3.1 The time complexity classes DTIME, $\mathcal{P}$, and $\mathcal{NP}$

Arora and Barak [2, Definitions 1.12, 1.13] define $\mathrm{DTIME}(T(n))$ as the set of all languages that can be decided in time $c \cdot T(n)$ for some $c > 0$ and $\mathcal{P} = \bigcup_{c\geq 1} \mathrm{DTIME}(n^c)$. However since $0^c = 0$ for $c \geq 1$, this means that for a language L to be in $\mathcal{P}$, the Turing machine deciding L must check the empty string in zero steps. For a Turing machine to halt in zero steps, it must start in the halting state, which limits its usefulness. Because of this technical issue we define $\mathrm{DTIME}(T(n))$ as the set of all languages that can be decided with a running time in $O(T(n))$, which seems a common enough alternative [11, 12, 1].

Languages are sets of strings, and deciding a language means computing its characteristic function.

type-synonym *language* = *string set*

definition *characteristic* :: *language* ⇒ (*string* ⇒ *string*) **where**
  characteristic L ≡ (λ*x*. [*x* ∈ *L*])

definition *DTIME* :: (*nat* ⇒ *nat*) ⇒ *language set* **where**
  DTIME T ≡ {*L*. ∃ *k G M T'*.
    turing-machine k G M ∧
    big-oh T' T ∧
    computes-in-time k M (*characteristic L*) *T'*}

definition *complexity-class-P* :: *language set* (‹$\mathcal{P}$›) **where**
  $\mathcal{P}$ ≡ ⋃ *c*∈{*1*..}. *DTIME* (λ*n*. *n* ^ *c*)

A language L is in $\mathcal{NP}$ if there is a polynomial p and a polynomial-time Turing machine, called the *verifier*, such that for all strings $x \in \{\mathbb{O}, \mathbb{I}\}^*$,

$$x \in L \longleftrightarrow \exists u \in \{\mathbb{O}, \mathbb{I}\}^{p(|x|)} : M(\langle x, u\rangle) = \mathbb{I}.$$

The string u does not seem to have a name in general, but in case the verifier outputs $\mathbb{I}$ on input $\langle x, u\rangle$ it is called a *certificate* for x [2, Definition 2.1].

definition *complexity-class-NP* :: *language set* (‹$\mathcal{NP}$›) **where**
  $\mathcal{NP}$ ≡ {*L*. ∃ *k G M p T fverify*.
    turing-machine k G M ∧
    polynomial p ∧

$big\text{-}oh\text{-}poly\ T\ \wedge$
$computes\text{-}in\text{-}time\ k\ M\ fverify\ T\ \wedge$
$(\forall x.\ x \in L \longleftrightarrow (\exists u.\ length\ u = p\ (length\ x) \wedge fverify\ \langle x,\ u\rangle = [\mathbb{I}]))\}$

The definition of $\mathcal{NP}$ is the one place where we need an actual polynomial function, namely p, rather than a function that is merely bounded by a polynomial. This raises the question as to the definition of a polynomial function. Arora and Barak [2] do not seem to give a definition in the context of $\mathcal{NP}$ but explicitly state that polynomial functions are mappings $\mathbb{N} \to \mathbb{N}$. Presumably they also have the form $f(n) = \sum_{i=0}^{d} a_i \cdot n^i$, as polynomials do. We have filled in the gap in this definition in Section 2.1.4 by letting the coefficients a_i be natural numbers.

Regardless of whether this is the meaning intended by Arora and Barak, the choice is justified because with it the verifier-based definition of $\mathcal{NP}$ is equivalent to the original definition via nondeterministic Turing machines (NTMs). In the usual equivalence proof (for example, Arora and Barak [2, Theorem 2.6]) a verifier TM and an NTM are constructed.

For the one direction, if a language is decided by a polynomial-time NTM then a verifier TM can be constructed that simulates the NTM on input x by using the bits in the string u for the nondeterministic choices. The strings u have the length $p(|x|)$. So for this construction to work, there must be a polynomial p that bounds the running time of the NTM. Clearly, a polynomial function with natural coefficients exists with that property.

For the other direction, assume a language has a verifier TM where p is a polynomial function with natural coefficients. An NTM deciding this language receives x as input, then "guesses" a string u of length $p(|x|)$, and then runs the verifier on the pair $\langle x, u\rangle$. For this NTM to run in polynomial time, p must be computable in time polynomial in $|x|$. We have shown this to be the case in lemma *transforms-tm-polynomialI* in Section 2.7.8.

A language L_1 is polynomial-time many-one reducible to a language L_2 if there is a polynomial-time computable function f_{reduce} such that for all x, $x \in L_1$ iff. $f_{reduce}(x) \in L_2$.

definition *reducible* (**infix** ‹$\leq_p$› *50*) **where**
  $L_1 \leq_p L_2 \equiv \exists k\ G\ M\ T\ freduce.$
    $turing\text{-}machine\ k\ G\ M\ \wedge$
    $big\text{-}oh\text{-}poly\ T\ \wedge$
    $computes\text{-}in\text{-}time\ k\ M\ freduce\ T\ \wedge$
    $(\forall x.\ x \in L_1 \longleftrightarrow freduce\ x \in L_2)$

abbreviation *NP-hard* :: *language* $\Rightarrow$ *bool* **where**
  $NP\text{-}hard\ L \equiv \forall L' \in \mathcal{NP}.\ L' \leq_p L$

definition *NP-complete* :: *language* $\Rightarrow$ *bool* **where**
  $NP\text{-}complete\ L \equiv L \in \mathcal{NP} \wedge NP\text{-}hard\ L$

Requiring $c \geq 1$ in the definition of $\mathcal{P}$ is not essential:

lemma *in-P-iff*: $L \in \mathcal{P} \longleftrightarrow (\exists c.\ L \in DTIME\ (\lambda n.\ n\ \hat{}\ c))$
proof
  assume $L \in \mathcal{P}$
  then show $\exists c.\ L \in DTIME\ (\lambda n.\ n\ \hat{}\ c)$
    unfolding *complexity-class-P-def* **using** *Suc-le-eq* **by** *auto*
next
  assume $\exists c.\ L \in DTIME\ (\lambda n.\ n\ \hat{}\ c)$
  then obtain c **where** $L \in DTIME\ (\lambda n.\ n\ \hat{}\ c)$
    by *auto*
  then obtain $k\ G\ M\ T$ **where** M:
    $turing\text{-}machine\ k\ G\ M$
    $big\text{-}oh\ T\ (\lambda n.\ n\ \hat{}\ c)$
    $computes\text{-}in\text{-}time\ k\ M\ (characteristic\ L)\ T$
    using *DTIME-def* **by** *auto*
  moreover have $big\text{-}oh\ T\ (\lambda n.\ n\ \hat{}\ Suc\ c)$
  proof −
    obtain $c0\ n0$ :: *nat* **where** $c0n0$: $\forall n{>}n0.\ T\ n \leq c0 * n\ \hat{}\ c$
      using $M(2)$ *big-oh-def* **by** *auto*
    have $\forall n{>}n0.\ T\ n \leq c0 * n\ \hat{}\ Suc\ c$
    proof *standard+*

```
      fix n assume n0 < n
      then have n ^ c ≤ n ^ Suc c
        using pow-mono by simp
      then show T n ≤ c0 * n ^ Suc c
        using c0n0 by (metis ‹n0 < n› add.commute le-Suc-ex mult-le-mono2 trans-le-add2)
    qed
    then show ?thesis
      using big-oh-def by auto
  qed
  ultimately have ∃c>0. L ∈ DTIME (λn. n ^ c)
    using DTIME-def by blast
  then show L ∈ 𝒫
    unfolding complexity-class-P-def by auto
qed
```

3.2 Restricting verifiers to one-bit output

The verifier Turing machine in the definition of $\mathcal{NP}$ can output any symbol sequence. In this section we restrict it to outputting only the symbol sequence **1** or **0**. This is equivalent to the definition because it is easy to check if a symbol sequence differs from **1** and if so change it to **0**, as we show below.

The advantage of this restriction is that if we can make the TM halt with the output tape head on cell number 1, the output tape symbol read in the final step equals the output of the TM. We will exploit this in Chapter 6, where we show how to reduce any language $L \in \mathcal{NP}$ to `SAT`.

The next Turing machine checks if the symbol sequence on tape j differs from the one-symbol sequence **1** and if so turns it into **0**. It thus ensures that the tape contains only one bit symbol.

```
definition tm-make-bit :: tapeidx ⇒ machine where
  tm-make-bit j ≡
    tm-cr j ;;
    IF λrs. rs ! j = 1 THEN
      tm-right j ;;
      IF λrs. rs ! j = □ THEN
        []
      ELSE
        tm-set j [0]
      ENDIF
    ELSE
      tm-set j [0]
    ENDIF

lemma tm-make-bit-tm:
  assumes G ≥ 4 and 0 < j and j < k
  shows turing-machine k G (tm-make-bit j)
  unfolding tm-make-bit-def
  using assms tm-right-tm tm-set-tm tm-cr-tm Nil-tm turing-machine-branch-turing-machine
  by simp

locale turing-machine-make-bit =
  fixes j :: tapeidx
begin

definition tm1 ≡ tm-cr j
definition tmT1 ≡ tm-right j
definition tmT12 ≡ IF λrs. rs ! j = □ THEN [] ELSE tm-set j [0] ENDIF
definition tmT2 ≡ tmT1 ;; tmT12
definition tm12 ≡ IF λrs. rs ! j = 1 THEN tmT2 ELSE tm-set j [0] ENDIF
definition tm2 ≡ tm1 ;; tm12

lemma tm2-eq-tm-make-bit: tm2 ≡ tm-make-bit j
  unfolding tm-make-bit-def tm2-def tm12-def tmT2-def tmT12-def tmT1-def tm1-def by simp
```

context
  fixes *tps0* :: *tape list* **and** *zs* :: *symbol list*
  assumes *jk*: $j < length\ tps0$
    and *zs*: *proper-symbols zs*
  assumes *tps0*: $tps0 ::: j = \lfloor zs \rfloor$
begin

lemma *clean*: *clean-tape* (*tps0* ! *j*)
  using *zs tps0 contents-clean-tape′* **by** *simp*

definition $tps1 \equiv tps0[j := (\lfloor zs \rfloor,\ 1)]$

lemma *tm1* [*transforms-intros*]:
  assumes $ttt = tps0 :\#: j + 2$
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def* **by** (*tform tps*: *assms jk clean tps0 tps1-def*)

definition $tpsT1 \equiv tps0[j := (\lfloor zs \rfloor,\ 2)]$

lemma *tmT1* [*transforms-intros*]:
  assumes $ttt = 1$
  shows *transforms tmT1 tps1 ttt tpsT1*
  unfolding *tmT1-def*
proof (*tform tps*: *assms tps1-def jk*)
  show $tpsT1 = tps1[j := tps1\ !\ j\ |{+}|\ 1]$
    using *tps1-def tpsT1-def jk*
    by (*metis Suc-1 fst-conv list-update-overwrite nth-list-update-eq plus-1-eq-Suc snd-conv*)
qed

definition $tpsT2 \equiv tps0$
  $[j := if\ length\ zs \leq 1\ then\ (\lfloor zs \rfloor,\ 2)\ else\ (\lfloor [\mathbf{0}] \rfloor,\ 1)]$

lemma *tmT12* [*transforms-intros*]:
  assumes $ttt = 14 + 2 * length\ zs$
  shows *transforms tmT12 tpsT1 ttt tpsT2*
  unfolding *tmT12-def*
proof (*tform tps*: *assms tpsT1-def tps0 jk zs*)
  show $8 + tpsT1 :\#: j + 2 * length\ zs + Suc\ (2 * length\ [\mathbf{0}]) + 1 \leq ttt$
    using *tpsT1-def jk assms* **by** *simp*
  have $read\ tpsT1\ !\ j = \lfloor zs \rfloor\ 2$
    using *tpsT1-def jk tapes-at-read′* **by** (*metis fst-conv length-list-update nth-list-update-eq snd-conv*)
  moreover have $\lfloor zs \rfloor\ 2 = \Box \longleftrightarrow length\ zs \leq 1$
    using *zs contents-def*
    by (*metis Suc-1 diff-Suc-1 less-imp-le-nat linorder-le-less-linear not-less-eq-eq zero-neq-numeral*)
  ultimately have $read\ tpsT1\ !\ j = \Box \longleftrightarrow length\ zs \leq 1$
    by *simp*
  then show
      $read\ tpsT1\ !\ j \neq \Box \Longrightarrow tpsT2 = tpsT1[j := (\lfloor [\mathbf{0}] \rfloor,\ 1)]$
      $read\ tpsT1\ !\ j = \Box \Longrightarrow tpsT2 = tpsT1$
    using *tpsT1-def tpsT2-def jk* **by** *simp-all*
qed

lemma *tmT2* [*transforms-intros*]:
  assumes $ttt = 15 + 2 * length\ zs$
  shows *transforms tmT2 tps1 ttt tpsT2*
  unfolding *tmT2-def* **by** (*tform time*: *assms*)

definition $tps2 \equiv tps0$
  $[j := if\ zs = [\mathbf{1}]\ then\ (\lfloor zs \rfloor,\ 2)\ else\ (\lfloor [\mathbf{0}] \rfloor,\ 1)]$

lemma *tm12* [*transforms-intros*]:
  assumes $ttt = 17 + 2 * length\ zs$
  shows *transforms tm12 tps1 ttt tps2*

```
  unfolding tm12-def
proof (tform tps: assms tps0 jk zs tps1-def)
  have read tps1 ! j = ⌊zs⌋ 1
    using tps1-def jk tapes-at-read' by (metis fst-conv length-list-update nth-list-update-eq snd-conv)
  then have ∗: read tps1 ! j = 𝟭 ⟷ ⌊zs⌋ 1 = 𝟭
    by simp
  show read tps1 ! j ≠ 𝟭 ⟹ tps2 = tps1[j := (⌊[𝟬]⌋, 1)]
    using ∗ tps2-def tps1-def by auto
  show tps2 = tpsT2 if read tps1 ! j = 𝟭
  proof (cases zs = [𝟭])
    case True
    then show ?thesis
      using ∗ tps2-def tpsT2-def by simp
  next
    case False
    then have ⌊zs⌋ 1 = 𝟭
      using ∗ that by simp
    then have length zs > 1
      using False contents-def contents-outofbounds
     by (metis One-nat-def Suc-length-conv diff-Suc-1 length-0-conv linorder-neqE-nat nth-Cons-0 zero-neq-numeral)
    then show ?thesis
      using ∗ tps2-def tpsT2-def by auto
  qed
  show 8 + tps1 :#: j + 2 ∗ length zs + Suc (2 ∗ length [𝟬]) + 1 ≤ ttt
    using tps1-def jk assms by simp
qed

lemma tm2:
  assumes ttt = 19 + 2 ∗ length zs + tps0 :#: j
  shows transforms tm2 tps0 ttt tps2
  unfolding tm2-def by (tform tps: assms tps0 jk zs tps1-def)

end

end

lemma transforms-tm-make-bitI [transforms-intros]:
  fixes j :: tapeidx
  fixes tps tps' :: tape list and zs :: symbol list and ttt :: nat
  assumes j < length tps and proper-symbols zs
  assumes tps ::: j = ⌊zs⌋
  assumes ttt = 19 + 2 ∗ length zs + tps :#: j
  assumes tps' = tps
    [j := if zs = [𝟭] then (⌊zs⌋, 2) else (⌊[𝟬]⌋, 1)]
  shows transforms (tm-make-bit j) tps ttt tps'
proof −
  interpret loc: turing-machine-make-bit j .
  show ?thesis
    using assms loc.tps2-def loc.tm2 loc.tm2-eq-tm-make-bit by simp
qed

lemma output-length-le-time:
  assumes turing-machine k G M
    and tps ::: 1 = ⌊zs⌋
    and proper-symbols zs
    and transforms M (snd (start-config k xs)) t tps
  shows length zs ≤ t
proof −
  have 1: execute M (start-config k xs) t = (length M, tps)
    using assms transforms-def transits-def
    by (metis (no-types, lifting) execute-after-halting-ge fst-conv start-config-def sndI)
  moreover have k > 1
    using assms(1) turing-machine-def by simp
```

```
  ultimately have ((execute M (start-config k xs) t) <:> 1) i = □ if i > t for i
    using assms blank-after-time that by (meson zero-less-one)
  then have (tps ::: 1) i = □ if i > t for i
    using 1 that by simp
  then have *: ⌊zs⌋ i = □ if i > t for i
    using assms(2) that by simp
  show ?thesis
  proof (rule ccontr)
    assume ¬ length zs ≤ t
    then have length zs > t
      by simp
    then have ⌊zs⌋ (Suc t) ≠ □
      using contents-inbounds assms(3) contents-def proper-symbols-ne0 by simp
    then show False
      using * by simp
  qed
qed
```

This is the alternative definition of $\mathcal{NP}$, which restricts the verifier to output only strings of length one:

```
lemma NP-output-len-1:
  𝒩𝒫 = {L. ∃ k G M p T fverify.
    turing-machine k G M ∧
    polynomial p ∧
    big-oh-poly T ∧
    computes-in-time k M fverify T ∧
    (∀ y. length (fverify y) = 1) ∧
    (∀ x. x ∈ L ⟷ (∃ u. length u = p (length x) ∧ fverify ⟨x, u⟩ = [𝕀]))}
  (is - = ?M)
proof
  show ?M ⊆ 𝒩𝒫
    using complexity-class-NP-def by fast
  define Q where Q = (λL k G M p T fverify.
    turing-machine k G M ∧
    polynomial p ∧
    big-oh-poly T ∧
    computes-in-time k M fverify T ∧
    (∀ x. (x ∈ L) = (∃ u. length u = p (length x) ∧ fverify ⟨x, u⟩ = [𝕀])))
  have alt: complexity-class-NP = {L::language. ∃ k G M p T fverify. Q L k G M p T fverify}
    unfolding complexity-class-NP-def Q-def by simp
  show 𝒩𝒫 ⊆ ?M
  proof
    fix L assume L ∈ 𝒩𝒫
    then obtain k G M p T fverify where Q L k G M p T fverify
      using alt by blast
    then have ex:
      turing-machine k G M ∧
       polynomial p ∧
       big-oh-poly T ∧
       computes-in-time k M fverify T ∧
       (∀ x. (x ∈ L) = (∃ u. length u = p (length x) ∧ fverify ⟨x, u⟩ = [𝕀]))
      using Q-def by simp

    have k ≥ 2 and G ≥ 4
      using ex turing-machine-def by simp-all

    define M' where M' = M ;; tm-make-bit 1
    define f' where f' = (λy. if fverify y = [𝕀] then [𝕀] else [𝕆])
    define T' where T' = (λn. 19 + 4 * T n)

    have turing-machine k G M'
      unfolding M'-def using ‹2 ≤ k› ‹4 ≤ G› ex tm-make-bit-tm by simp
    moreover have polynomial p
      using ex by simp
```

moreover have *big-oh-poly T′*
  using *T′-def ex big-oh-poly-const big-oh-poly-prod big-oh-poly-sum* **by** *simp*
moreover have *computes-in-time k M′ f′ T′*
proof
  fix *y*
  let *?cfg = start-config k (string-to-symbols y)*
  obtain *tps* **where**
    1: tps ::: 1 = string-to-contents (fverify y) **and**
    2: transforms M (snd ?cfg) (T (length y)) tps
    using *ex computes-in-timeD* **by** *blast*
  have *len-tps: length tps ≥ 2*
    by *(smt (verit) 2 ‹2 ≤ k› ex execute-num-tapes start-config-length less-le-trans numeral-2-eq-2 prod.sel(2) transforms-def transits-def zero-less-Suc)*
  define *zs* **where** *zs = string-to-symbols (fverify y)*
  then have *zs: tps ::: 1 = ⌊zs⌋ proper-symbols zs*
    using *1* **by** *auto*
  have *transforms-MI [transforms-intros]: transforms M (snd ?cfg) (T (length y)) tps*
    using *2* **by** *simp*
  define *tps′* **where**
    *tps′ = tps[1 := if zs = [**1**] then (⌊zs⌋, 2) else (⌊[**0**]⌋, 1)]*

  have *transforms M′ (snd ?cfg) (T (length y) + (19 + (tps :#: Suc 0 + 2 ∗ length zs))) tps′*
    unfolding *M′-def* **by** *(tform tps: zs len-tps tps′-def)*
  moreover have *T (length y) + (19 + (tps :#: Suc 0 + 2 ∗ length zs)) ≤ T′ (length y)*
  proof −
    have *tps :#: Suc 0 ≤ T (length y)*
      using *2 transforms-def transits-def start-config-def ex head-pos-le-time ‹2 ≤ k›*
      by *(smt (verit, ccfv-threshold) One-nat-def Suc-1 Suc-le-lessD leD linorder-le-less-linear order-less-le-trans prod.sel(2))*
    moreover have *length zs ≤ T (length y)*
      using *zs 2 ex output-length-le-time* **by** *auto*
    ultimately show *?thesis*
      using *T′-def 1 2* **by** *simp*
  qed
  ultimately have *∗: transforms M′ (snd ?cfg) (T′ (length y)) tps′*
    using *transforms-monotone* **by** *simp*

  have *tps′ ::: 1 = (if zs = [**1**] then tps ::: 1 else ⌊[**0**]⌋)*
    using *tps′-def len-tps zs(1)* **by** *simp*
  also have *... = (if zs = [**1**] then ⌊[**1**]⌋ else ⌊[**0**]⌋)*
    using *zs(1)* **by** *simp*
  also have *... = (if string-to-symbols (fverify y) = [3] then ⌊[3]⌋ else ⌊[2]⌋)*
    using *zs-def* **by** *simp*
  also have *... = (if fverify y = [𝕀] then ⌊[**1**]⌋ else ⌊[**0**]⌋)*
    by *auto*
  also have *... = (if fverify y = [𝕀] then string-to-contents [𝕀] else string-to-contents [𝕆])*
    by *auto*
  also have *... = string-to-contents (if fverify y = [𝕀] then [𝕀] else [𝕆])*
    by *simp*
  also have *... = string-to-contents (f′ y)*
    using *f′-def* **by** *auto*
  finally have *tps′ ::: 1 = string-to-contents (f′ y)* **.**

  then show *∃ tps′. tps′ ::: 1 = string-to-contents (f′ y) ∧*
      transforms M′ (snd ?cfg) (T′ (length y)) tps′
    using *∗* **by** *auto*
qed
moreover have *∀ y. length (f′ y) = 1*
  using *f′-def* **by** *simp*
moreover have *(∀ x. x ∈ L ⟷ (∃ u. length u = p (length x) ∧ f′ ⟨x, u⟩ = [𝕀]))*
  using *ex f′-def* **by** *auto*
ultimately show *L ∈ ?M*
  by *blast*

qed
qed

3.3 $\mathcal{P}$ is a subset of $\mathcal{NP}$

Let $L \in \mathcal{P}$ be a language and M a Turing machine that decides L in polynomial time. To show $L \in \mathcal{NP}$ we could use a TM that extracts the first element from the input $\langle x, u\rangle$ and runs M on x. We do not have to construct such a TM explicitly because we have shown that the extraction of the first pair element is computable in polynomial time (lemma *tm-first-computes*), and by assumption the characteristic function of L is computable in polynomial time, too. The composition of these two functions is the verifier function required by the definition of $\mathcal{NP}$. And by lemma *computes-composed-poly* the composition of polynomial-time functions is polynomial-time, too.

theorem *P-subseteq-NP*: $\mathcal{P} \subseteq \mathcal{NP}$
proof
  fix L :: *language*
  assume $L \in \mathcal{P}$
  then obtain c **where** c: $L \in$ *DTIME* ($\lambda n.\ n$ ^ c)
    using *complexity-class-P-def* **by** *auto*
  then obtain $k\ G\ M\ T'$ **where** M:
    turing-machine k G M
    computes-in-time k M (*characteristic L*) T'
    big-oh T' ($\lambda n.\ n$ ^ c)
    using *DTIME-def* **by** *auto*
  then have *4*: *big-oh-poly* T'
    using *big-oh-poly-def* **by** *auto*

  define f **where** $f = (\lambda x.$ *symbols-to-string* (*first* (*bindecode* (*string-to-symbols x*))))
  define T :: *nat* $\Rightarrow$ *nat* **where** $T = (\lambda n.\ 9 + 4 * n)$
  have *1*: *turing-machine 3 6 tm-first*
    by (*simp add*: *tm-first-tm*)
  have *2*: *computes-in-time 3 tm-first f T*
    using *f-def T-def tm-first-computes* **by** *simp*
  have *3*: *big-oh-poly T*
  proof −
    have *big-oh-poly* ($\lambda n.\ 9$)
      using *big-oh-poly-const* **by** *simp*
    moreover have *big-oh-poly* ($\lambda n.\ 4 * n$)
      using *big-oh-poly-const big-oh-poly-prod big-oh-poly-poly*[*of 1*] **by** *simp*
    ultimately show *?thesis*
      using *big-oh-poly-sum T-def* **by** *simp*
  qed

  define *fverify* **where** *fverify* = *characteristic L* $\circ$ *f*
  then have $*$: $\exists\, T\ k\ G\ M.$ *big-oh-poly T* $\wedge$ *turing-machine k G M* $\wedge$ *computes-in-time k M fverify T*
    using *M 1 2 3 4 computes-composed-poly* **by** *simp*

  then have *fverify*: *fverify* $\langle x,\ u\rangle = [x \in L]$ **for** *x u*
    using *f-def first-pair symbols-to-string-to-symbols fverify-def characteristic-def*
    by *simp*

  define p :: *nat* $\Rightarrow$ *nat* **where** $p = (\lambda\text{-}.\ 0)$
  then have *polynomial p*
    using *const-polynomial* **by** *simp*
  moreover have $\forall\, x.\ x \in L \longleftrightarrow (\exists\, u.$ *length* $u = p$ (*length x*) $\wedge$ *fverify* $\langle x,\ u\rangle = [\mathbb{1}])$
    using *fverify p-def* **by** *simp*
  ultimately show $L \in \mathcal{NP}$
    using $*$ *complexity-class-NP-def* **by** *fast*
qed

3.4 More about $\mathcal{P}$, $\mathcal{NP}$, and reducibility

We prove some low-hanging fruits about the concepts introduced in this chapter. None of the results are needed to show the Cook-Levin theorem.

A language can be reduced to itself by the identity function. Hence reducibility is a reflexive relation.

```
lemma reducible-refl: L ≤p L
proof −
  let ?M = tm-id
  let ?T = λn. Suc (Suc n)
  have turing-machine 2 4 ?M
    using tm-id-tm by simp
  moreover have big-oh-poly ?T
  proof −
    have big-oh-poly (λn. n + 2)
      using big-oh-poly-const big-oh-poly-id big-oh-poly-sum by blast
    then show ?thesis
      by simp
  qed
  moreover have computes-in-time 2 ?M id ?T
    using computes-id by simp
  moreover have ∀ x. x ∈ L ⟷ id x ∈ L
    by simp
  ultimately show L ≤p L
    using reducible-def by metis
qed
```

Reducibility is also transitive. If $L_1 \leq_p L_2$ by a TM M_1 and $L_2 \leq_p L_3$ by a TM M_2 we merely have to run M_2 on the output of M_1 to show that $L_1 \leq_p L_3$. Again this is merely the composition of two polynomial-time computable functions.

```
lemma reducible-trans:
  assumes L1 ≤p L2 and L2 ≤p L3
  shows L1 ≤p L3
proof −
  obtain k1 G1 M1 T1 f1 where 1:
     turing-machine k1 G1 M1 ∧
      big-oh-poly T1 ∧
      computes-in-time k1 M1 f1 T1 ∧
      (∀ x. x ∈ L1 ⟷ f1 x ∈ L2)
    using assms(1) reducible-def by metis
  moreover obtain k2 G2 M2 T2 f2 where 2:
     turing-machine k2 G2 M2 ∧
      big-oh-poly T2 ∧
      computes-in-time k2 M2 f2 T2 ∧
      (∀ x. x ∈ L2 ⟷ f2 x ∈ L3)
    using assms(2) reducible-def by metis
  ultimately obtain T k G M where
      big-oh-poly T ∧ turing-machine k G M ∧ computes-in-time k M (f2 ∘ f1) T
    using computes-composed-poly by metis
  moreover have ∀ x. x ∈ L1 ⟷ f2 (f1 x) ∈ L3
    using 1 2 by simp
  ultimately show ?thesis
    using reducible-def by fastforce
qed
```

The usual way to show that a language is $\mathcal{NP}$-hard is to reduce another $\mathcal{NP}$-hard language to it.

```
lemma ex-reducible-imp-NP-hard:
  assumes NP-hard L′ and L′ ≤p L
  shows NP-hard L
  using reducible-trans assms by auto
```

The converse is also true because reducibility is a reflexive relation.

lemma *NP-hard-iff-reducible*: *NP-hard* $L \longleftrightarrow (\exists L'.$ *NP-hard* $L' \wedge L' \leq_p L)$
proof
  show *NP-hard* $L \Longrightarrow \exists L'.$ *NP-hard* $L' \wedge L' \leq_p L$
    using *reducible-refl* **by** *auto*
  show $\exists L'.$ *NP-hard* $L' \wedge L' \leq_p L \Longrightarrow$ *NP-hard* L
    using *ex-reducible-imp-NP-hard* **by** *blast*
qed

lemma *NP-complete-reducible*:
  assumes *NP-complete* L' **and** $L \in \mathcal{NP}$ **and** $L' \leq_p L$
  shows *NP-complete* L
  using *assms NP-complete-def reducible-trans* **by** *blast*

In a sense the complexity class $\mathcal{P}$ is closed under reduction.

lemma *P-closed-reduction*:
  assumes $L \in \mathcal{P}$ **and** $L' \leq_p L$
  shows $L' \in \mathcal{P}$
proof −
  obtain c **where** c: $L \in$ *DTIME* $(\lambda n.\ n \mathbin{\hat{}} c)$
    using *assms*(*1*) *complexity-class-P-def* **by** *auto*
  then obtain k G M T **where** M:
    turing-machine k G M
    computes-in-time k M (*characteristic* L) T
    big-oh T $(\lambda n.\ n \mathbin{\hat{}} c)$
    using *DTIME-def* **by** *auto*
  then have T: *big-oh-poly* T
    using *big-oh-poly-def* **by** *auto*

  obtain k' G' M' T' *freduce* **where** M':
    turing-machine k' G' M'
    big-oh-poly T'
    computes-in-time k' M' *freduce* T'
    $(\forall x.\ x \in L' \longleftrightarrow$ *freduce* $x \in L)$
    using *reducible-def assms*(*2*) **by** *auto*

  obtain *T2 k2 G2 M2* **where** *M2*:
    big-oh-poly T2
    turing-machine k2 G2 M2
    computes-in-time k2 M2 (*characteristic* $L \circ$ *freduce*) *T2*
    using M T M' *computes-composed-poly* **by** *metis*

  obtain d **where** d: *big-oh T2* $(\lambda n.\ n \mathbin{\hat{}} d)$
    using *big-oh-poly-def M2*(*1*) **by** *auto*

  have *characteristic* $L \circ$ *freduce* $=$ *characteristic* L'
    using *characteristic-def* M'(*4*) **by** *auto*
  then have $\exists k\ G\ M\ T'.$ *turing-machine* k G M $\wedge$ *big-oh* T' $(\lambda n.\ n \mathbin{\hat{}} d)$ $\wedge$ *computes-in-time* k M (*characteristic* L') T'
    using *M2* d **by** *auto*
  then have $L' \in$ *DTIME* $(\lambda n.\ n \mathbin{\hat{}} d)$
    using *DTIME-def* **by** *simp*
  then show *?thesis*
    using *in-P-iff* **by** *auto*
qed

The next lemmas are items 2 and 3 of Theorem 2.8 of the textbook [2]. Item 1 is the transitivity of the reduction, already proved in lemma *reducible-trans*.

lemma *P-eq-NP*:
  assumes *NP-hard* L **and** $L \in \mathcal{P}$
  shows $\mathcal{P} = \mathcal{NP}$
  using *assms P-closed-reduction P-subseteq-NP* **by** *auto*

lemma *NP-complete-imp*:

assumes *NP-complete L*
shows $L \in \mathcal{P} \longleftrightarrow \mathcal{P} = \mathcal{NP}$
using *assms NP-complete-def P-eq-NP* **by** *auto*

end

Chapter 4

Satisfiability

theory *Satisfiability*
  imports *Wellformed NP*
begin

This chapter introduces the language `SAT` and shows that it is in $\mathcal{NP}$, which constitutes the easier part of the Cook-Levin theorem. The other part, the $\mathcal{NP}$-hardness of `SAT`, is what all the following chapters are concerned with.

We first introduce Boolean formulas in conjunctive normal form and the concept of satisfiability. Then we define a way to represent such formulas as bit strings, leading to the definition of the language `SAT` as a set of strings (Section 4.1).

For the proof that `SAT` is in $\mathcal{NP}$, we construct a Turing machine that, given a CNF formula and a string representing a variable assignment, outputs **1** iff. the assignment satisfies the formula. The TM will run in polynomial time, and there are always assignments polynomial (in fact, linear) in the length of the formula (Section 4.2).

4.1 The language `SAT`

`SAT` is the language of all strings representing satisfiable Boolean formulas in conjunctive normal form (CNF). This section introduces a minimal version of Boolean formulas in conjunctive normal form, including the concepts of assignments and satisfiability.

4.1.1 CNF formulas and satisfiability

Arora and Barak [2, p. 44] define Boolean formulas in general as expressions over $\wedge$, $\vee$, $\neg$, parentheses, and variables $v_1, v_2, \dots$ in the usual way. Boolean formulas in conjunctive normal form are defined as $\bigwedge_i \left(\bigvee_j v_{i_j}\right)$, where the v_{i_j} are literals. This definition does not seem to allow for empty clauses. Also whether the "empty CNF formula" exists is somewhat doubtful. Nevertheless, our formalization allows for both empty clauses and the empty CNF formula, because this enables us to represent CNF formulas as lists of clauses and clauses as lists of literals without having to somehow forbid the empty list. This seems to be a popular approach for formalizing CNF formulas in the context of `SAT` and $\mathcal{NP}$ [7, 14].

We identify a variable v_i with its index i, which can be any natural number. A *literal* can either be positive or negative, representing a variable or negated variable, respectively.

datatype *literal* = *Neg nat* | *Pos nat*

type-synonym *clause* = *literal list*

type-synonym *formula* = *clause list*

An *assignment* maps all variables, given by their index, to a Boolean:

type-synonym *assignment* = *nat* $\Rightarrow$ *bool*

abbreviation *satisfies-literal* :: *assignment* $\Rightarrow$ *literal* $\Rightarrow$ *bool* **where**
satisfies-literal α x $\equiv$ *case* x *of Neg* $n \Rightarrow \neg\ \alpha\ n$ | *Pos* $n \Rightarrow \alpha\ n$

definition *satisfies-clause* :: *assignment* $\Rightarrow$ *clause* $\Rightarrow$ *bool* **where**
satisfies-clause α $c \equiv \exists\, x{\in}$*set* c. *satisfies-literal* α x

definition *satisfies* :: *assignment* $\Rightarrow$ *formula* $\Rightarrow$ *bool* (**infix** ‹$\models$› *60*) **where**
$\alpha \models \varphi \equiv \forall\, c{\in}$*set* φ. *satisfies-clause* α c

As is customary, the empty clause is satisfied by no assignment, and the empty CNF formula is satisfied by every assignment.

proposition $\neg$ *satisfies-clause* α []
by (*simp add*: *satisfies-clause-def*)

proposition $\alpha \models$ []
by (*simp add*: *satisfies-def*)

lemma *satisfies-clause-take*:
assumes $i <$ *length clause*
shows *satisfies-clause* α (*take* (*Suc* i) *clause*) $\longleftrightarrow$
satisfies-clause α (*take* i *clause*) $\vee$ *satisfies-literal* α (*clause* ! i)
using *assms satisfies-clause-def* **by** (*auto simp add*: *take-Suc-conv-app-nth*)

lemma *satisfies-take*:
assumes $i <$ *length* φ
shows $\alpha \models$ *take* (*Suc* i) $\varphi \longleftrightarrow \alpha \models$ *take* i φ $\wedge$ *satisfies-clause* α (φ ! i)
using *assms satisfies-def* **by** (*auto simp add*: *take-Suc-conv-app-nth*)

lemma *satisfies-append*:
assumes $\alpha \models \varphi_1 @ \varphi_2$
shows $\alpha \models \varphi_1$ **and** $\alpha \models \varphi_2$
using *assms satisfies-def* **by** *simp-all*

lemma *satisfies-append'*:
assumes $\alpha \models \varphi_1$ **and** $\alpha \models \varphi_2$
shows $\alpha \models \varphi_1 @ \varphi_2$
using *assms satisfies-def* **by** *auto*

lemma *satisfies-concat-map*:
assumes $\alpha \models$ *concat* (*map f* [*0*..<k]) **and** $i < k$
shows $\alpha \models f\ i$
using *assms satisfies-def* **by** *simp*

lemma *satisfies-concat-map'*:
assumes $\bigwedge i.\ i < k \Longrightarrow \alpha \models f\ i$
shows $\alpha \models$ *concat* (*map f* [*0*..<k])
using *assms satisfies-def* **by** *simp*

The main ingredient for defining `SAT` is the concept of *satisfiable* CNF formula:

definition *satisfiable* :: *formula* $\Rightarrow$ *bool* **where**
satisfiable $\varphi \equiv \exists\,\alpha.\ \alpha \models \varphi$

The set of all variables used in a CNF formula is finite.

definition *variables* :: *formula* $\Rightarrow$ *nat set* **where**
variables $\varphi \equiv \{n.\ \exists\, c{\in}$*set* φ. *Neg* $n \in$ *set* c $\vee$ *Pos* $n \in$ *set* $c\}$

lemma *finite-variables*: *finite* (*variables* φ)
proof −
define *voc* :: *clause* $\Rightarrow$ *nat set* **where**
voc $c = \{n.$ *Neg* $n \in$ *set* c $\vee$ *Pos* $n \in$ *set* $c\}$ **for** c
let *?vocs* = *set* (*map voc* φ)

have *finite* (*voc* c) **for** c

proof (*induction c*)
  case *Nil*
  then show *?case*
    using *voc-def* **by** *simp*
next
  case (*Cons a c*)
  have $voc\ (a\ \#\ c) = \{n.\ Neg\ n \in set\ (a\ \#\ c) \vee Pos\ n \in set\ (a\ \#\ c)\}$
    using *voc-def* **by** *simp*
  also have ... $= \{n.\ Neg\ n \in set\ c \vee Neg\ n = a \vee Pos\ n \in set\ c \vee Pos\ n = a\}$
    by *auto*
  also have ... $= \{n.\ (Neg\ n \in set\ c \vee Pos\ n \in set\ c) \vee (Pos\ n = a \vee Neg\ n = a)\}$
    by *auto*
  also have ... $= \{n.\ Neg\ n \in set\ c \vee Pos\ n \in set\ c\} \cup \{n.\ Pos\ n = a \vee Neg\ n = a\}$
    by *auto*
  also have ... $= voc\ c \cup \{n.\ Pos\ n = a \vee Neg\ n = a\}$
    using *voc-def* **by** *simp*
  finally have $voc\ (a\ \#\ c) = voc\ c \cup \{n.\ Pos\ n = a \vee Neg\ n = a\}$.
  moreover have *finite* $\{n.\ Pos\ n = a \vee Neg\ n = a\}$
    using *finite-nat-set-iff-bounded* **by** *auto*
  ultimately show *?case*
    using *Cons* **by** *simp*
qed
moreover have *variables* $\varphi = \bigcup$ *?vocs*
  using *variables-def voc-def* **by** *auto*
moreover have *finite ?vocs*
  by *simp*
ultimately show *?thesis*
  by *simp*
qed

lemma *variables-append*: *variables* $(\varphi_1 \mathbin{@} \varphi_2) =$ *variables* $\varphi_1 \cup$ *variables* φ_2
  using *variables-def* **by** *auto*

Arora and Barak [2, Claim 2.13] define the *size* of a CNF formula as the numbr of $\wedge/\vee$ symbols. We use a slightly different definition, namely the number of literals:

definition *fsize* :: *formula* $\Rightarrow$ *nat* **where**
  fsize $\varphi \equiv$ *sum-list* (*map length* φ)

4.1.2 Predicates on assignments

Every CNF formula is satisfied by a set of assignments. Conversely, for certain sets of assignments we can construct CNF formulas satisfied by exactly these assignments. This will be helpful later when we construct formulas for reducing arbitrary languages to SAT (see Section 6).

Universality of CNF formulas

A set (represented by a predicate) F of assignments depends on the first ℓ variables iff. any two assignments that agree on the first ℓ variables are either both in the set or both outside of the set.

definition *depon* :: *nat* $\Rightarrow$ (*assignment* $\Rightarrow$ *bool*) $\Rightarrow$ *bool* **where**
  $depon\ l\ F \equiv \forall \alpha_1\ \alpha_2.\ (\forall i<l.\ \alpha_1\ i = \alpha_2\ i) \longrightarrow F\ \alpha_1 = F\ \alpha_2$

Lists of all strings of the same length:

fun *str-of-len* :: *nat* $\Rightarrow$ *string list* **where**
  str-of-len $0 = [[]]$ |
  str-of-len $(Suc\ l) = map\ ((\#)\ \mathbb{O})\ (str\text{-}of\text{-}len\ l) \mathbin{@} map\ ((\#)\ \mathbb{I})\ (str\text{-}of\text{-}len\ l)$

lemma *length-str-of-len*: $length\ (str\text{-}of\text{-}len\ l) = 2\ \widehat{}\ l$
  by (*induction l*) *simp-all*

lemma *in-str-of-len-length*: $xs \in set\ (str\text{-}of\text{-}len\ l) \Longrightarrow length\ xs = l$
  by (*induction l arbitrary*: *xs*) *auto*

lemma *length-in-str-of-len*: *length xs* = *l* $\Longrightarrow$ *xs* $\in$ *set* (*str-of-len l*)
proof (*induction l arbitrary*: *xs*)
  case *0*
  then show *?case*
    by *simp*
next
  case (*Suc l*)
  then obtain *y ys* **where** *xs* = *y* # *ys*
    by (*meson length-Suc-conv*)
  then have *length ys* = *l*
    using *Suc* **by** *simp*
  show *?case*
  proof (*cases y*)
    case *True*
    then have *xs* $\in$ *set* (*map* ((#) $\mathbb{I}$) (*str-of-len l*))
      using ‹*length ys* = *l*› *Suc* ‹*xs* = *y* # *ys*› **by** *simp*
    then show *?thesis*
      by *simp*
  next
    case *False*
    then have *xs* $\in$ *set* (*map* ((#) $\mathbb{O}$) (*str-of-len l*))
      using ‹*length ys* = *l*› *Suc* ‹*xs* = *y* # *ys*› **by** *simp*
    then show *?thesis*
      by *simp*
  qed
qed

A predicate F depending on the first ℓ variables $v_0, \dots, v_{\ell-1}$ can be regarded as a truth table over ℓ variables. The next lemma shows that for every such truth table there exists a CNF formula with at most 2^ℓ clauses and $\ell \cdot 2^\ell$ literals over the first ℓ variables. This is the well-known fact that every Boolean function (over ℓ variables) can be represented by a CNF formula [2, Claim 2.13].

lemma *depon-ex-formula*:
  assumes *depon l F*
  shows $\exists\,\varphi$.
    fsize $\varphi \le l * 2 \mathbin{\hat{}} l \wedge$
    length $\varphi \le 2 \mathbin{\hat{}} l \wedge$
    variables $\varphi \subseteq \{..<l\} \wedge$
    $(\forall\,\alpha.\ F\ \alpha = \alpha \models \varphi)$
proof −
  define *cl* **where** *cl* = (λ*v*. *map* (λ*i*. *if v* ! *i then Neg i else Pos i*) [*0*..<*l*])
  have *cl1*: *satisfies-clause a* (*cl v*) **if** *length v* = *l* **and** *v* $\neq$ *map a* [*0*..<*l*] **for** *v a*
  proof −
    obtain *i* **where** *i*: *i* < *l a i* $\neq$ *v* ! *i*
      using ‹*length v* = *l*› ‹*v* $\neq$ *map a* [*0*..<*l*]›
      by (*smt* (*verit*, *best*) *atLeastLessThan-iff map-eq-conv map-nth set-upt*)
    then have $*$: *cl v* ! *i* = (*if v* ! *i then Neg i else Pos i*)
      using *cl-def* **by** *simp*
    then have *case* (*cl v* ! *i*) *of Neg n* $\Rightarrow$ ¬ *a n* | *Pos n* $\Rightarrow$ *a n*
      using *i*(*2*) **by** *simp*
    then show *?thesis*
      using *cl-def* $*$ *that*(*1*) *satisfies-clause-def i*(*1*) **by** *fastforce*
  qed
  have *cl2*: *v* $\neq$ *map a* [*0*..<*l*] **if** *length v* = *l* **and** *satisfies-clause a* (*cl v*) **for** *v a*
  proof
    assume *assm*: *v* = *map a* [*0*..<*l*]
    from *that*(*2*) **have** $\exists$*x*$\in$*set* (*cl v*). *case x of Neg n* $\Rightarrow$ ¬ *a n* | *Pos n* $\Rightarrow$ *a n*
      using *satisfies-clause-def* **by** *simp*
    then obtain *i* **where** *i*: *i* < *l* **and** *case* (*cl v* ! *i*) *of Neg n* $\Rightarrow$ ¬ *a n* | *Pos n* $\Rightarrow$ *a n*
      using *cl-def* **by** *auto*
    then have *v* ! *i* $\neq$ *a i*
      using *cl-def* **by** *fastforce*
    then show *False*

    using *i assm* **by** *simp*
qed

have *filter-length-nth*: *f* (*vs* ! *j*) **if** *vs* = *filter f sol* **and** *j* < *length vs*
  for *vs sol* :: *'a list* **and** *f j*
  using *that nth-mem* **by** (*metis length-removeAll-less less-irrefl removeAll-filter-not*)

have *sum-list-map*: *sum-list* (*map g xs*) ≤ *k* ∗ *length xs* **if** ⋀*x*. *x* ∈ *set xs* ⟹ *g x* = *k*
  for *xs* :: *'a list* **and** *g k*
  using *that*
proof (*induction length xs arbitrary*: *xs*)
  case *0*
  then show *?case*
    by *simp*
next
  case (*Suc x*)
  then obtain *y ys* **where** *xs* = *y* # *ys*
    by (*metis length-Suc-conv*)
  then have *length ys* = *x*
    using *Suc* **by** *simp*
  have *y* ∈ *set xs*
    using ‹*xs* = *y* # *ys*› **by** *simp*
  have *sum-list* (*map g xs*) = *sum-list* (*map g* (*y* # *ys*))
    using ‹*xs* = *y* # *ys*› **by** *simp*
  also have ... = *g y* + *sum-list* (*map g ys*)
    by *simp*
  also have ... = *k* + *sum-list* (*map g ys*)
    using *Suc* ‹*y* ∈ *set xs*› **by** *simp*
  also have ... ≤ *k* + *k* ∗ *length ys*
    using *Suc* ‹*length ys* = *x*› ‹*xs* = *y* # *ys*› **by** *auto*
  also have ... = *k* ∗ *length xs*
    by (*metis Suc.hyps*(*2*) ‹*length ys* = *x*› *mult-Suc-right*)
  finally show *?case*
    by *simp*
qed

define *vs* **where**
  vs = *filter* (λ*v*. *F* (λ*i*. *if i* < *l then v* ! *i else False*) = *False*) (*str-of-len l*)
define φ **where** φ = *map cl vs*

have *a* ⊨ φ **if** *F a* **for** *a*
proof −
  define *v* **where** *v* = *map a* [*0*..<*l*]
  then have (λ*i*. *if i* < *l then v* ! *i else False*) *j* = *a j* **if** *j* < *l* **for** *j*
    by (*simp add*: *that*)
  then have ∗: *F* (λ*i*. *if i* < *l then v* ! *i else False*)
    using *assms*(*1*) *depon-def that* **by** (*smt* (*verit*, *ccfv-SIG*))
  have *satisfies-clause a c* **if** *c* ∈ *set* φ **for** *c*
  proof −
    obtain *j* **where** *j*: *c* = φ ! *j j* < *length* φ
      using φ*-def* ‹*c* ∈ *set* φ› **by** (*metis in-set-conv-nth*)
    then have *c* = *cl* (*vs* ! *j*)
      using φ*-def* **by** *simp*
    have *j* < *length vs*
      using φ*-def j* **by** *simp*
    then have *F* (λ*i*. *if i* < *l then* (*vs* ! *j*) ! *i else False*) = *False*
      using *vs-def filter-length-nth* **by** *blast*
    then have *vs* ! *j* ≠ *v*
      using ∗ **by** *auto*
    moreover have *length* (*vs* ! *j*) = *l*
      using *vs-def length-str-of-len* ‹*j* < *length vs*›
      by (*smt* (*verit*) *filter-eq-nths in-str-of-len-length notin-set-nthsI nth-mem*)
    ultimately have *satisfies-clause a* (*cl* (*vs* ! *j*))

```
      using v-def cl1 by simp
    then show ?thesis
      using ‹c = cl (vs ! j)› by simp
  qed
  then show ?thesis
    using satisfies-def by simp
qed
moreover have F α if α ⊨ φ for α
proof (rule ccontr)
  assume assm: ¬ F α
  define v where v = map α [0..<l]
  have *: F (λi. if i < l then v ! i else False) = False
  proof −
    have (λi. if i < l then v ! i else False) j = α j if j < l for j
      using v-def by (simp add: that)
    then show ?thesis
      using assm assms(1) depon-def by (smt (verit, best))
  qed
  have length v = l
    using v-def by simp
  then obtain j where j < length (str-of-len l) and v = str-of-len l ! j
    by (metis in-set-conv-nth length-in-str-of-len)
  then have v ∈ set vs
    using vs-def * by fastforce
  then have cl v ∈ set φ
    using φ-def by simp
  then have satisfies-clause α (cl v)
    using that satisfies-def by simp
  then have v ≠ map α [0..<l]
    using ‹length v = l› cl2 by simp
  then show False
    using v-def by simp
qed
ultimately have ∀ α. F α = α ⊨ φ
  by auto
moreover have fsize φ ≤ l ∗ 2 ^ l
proof −
  have length c = l if c ∈ set φ for c
    using that cl-def φ-def by auto
  then have fsize φ ≤ l ∗ length φ
    unfolding fsize-def using sum-list-map by auto
  also have ... ≤ l ∗ length (str-of-len l)
    using φ-def vs-def by simp
  also have ... = l ∗ 2 ^ l
    using length-str-of-len by simp
  finally show ?thesis .
qed
moreover have length φ ≤ 2 ^ l
proof −
  have length φ ≤ length (str-of-len l)
    using φ-def vs-def by simp
  also have ... = 2 ^ l
    using length-str-of-len by simp
  finally show ?thesis .
qed
moreover have variables φ ⊆ {..<l}
proof
  fix x assume x ∈ variables φ
  then obtain c where c: c ∈ set φ Neg x ∈ set c ∨ Pos x ∈ set c
    using variables-def by auto
  then obtain v where v: v ∈ set (str-of-len l) c = cl v
    using φ-def vs-def by auto
  then show x ∈ {..<l}
```

using *cl-def c* **by** *auto*
qed
ultimately show *?thesis*
by *auto*
qed

Substitutions of variables

We will sometimes consider CNF formulas over the first ℓ variables and derive other CNF formulas from them by substituting these variables. Such a substitution will be represented by a list σ of length at least ℓ, meaning that the variable v_i is replaced by $v_{\sigma(i)}$. We will call this operation on formulas *relabel*, and the corresponding one on literals *rename*:

fun *rename* :: *nat list* ⇒ *literal* ⇒ *literal* **where**
rename σ (*Neg i*) = *Neg* (σ ! *i*) |
rename σ (*Pos i*) = *Pos* (σ ! *i*)

definition *relabel* :: *nat list* ⇒ *formula* ⇒ *formula* **where**
relabel σ φ ≡ *map* (*map* (*rename* σ)) φ

lemma *fsize-relabel*: *fsize* (*relabel* σ φ) = *fsize* φ
using *relabel-def fsize-def* **by** (*metis length-concat length-map map-concat*)

A substitution σ can also be applied to an assignment and to a list of variable indices:

definition *remap* :: *nat list* ⇒ *assignment* ⇒ *assignment* **where**
remap σ α *i* ≡ *if i* < *length* σ *then* α (σ ! *i*) *else* α *i*

definition *reseq* :: *nat list* ⇒ *nat list* ⇒ *nat list* **where**
reseq σ *vs* ≡ *map* ((!) σ) *vs*

lemma *length-reseq* [*simp*]: *length* (*reseq* σ *vs*) = *length vs*
using *reseq-def* **by** *simp*

Relabeling a formula and remapping an assignment are equivalent in a sense.

lemma *satisfies-sigma*:
assumes *variables* φ ⊆ {..<*length* σ}
shows α ⊨ *relabel* σ φ ⟷ *remap* σ α ⊨ φ
proof
assume *sat*: α ⊨ *relabel* σ φ
have *satisfies-clause* (*remap* σ α) *c* **if** *c* ∈ *set* φ **for** *c*
proof −
obtain *i* **where** *i* < *length* φ φ ! *i* = *c*
by (*meson* ‹*c* ∈ *set* φ› *in-set-conv-nth*)
then have *satisfies-clause* α (*map* (*rename* σ) *c*)
(**is** *satisfies-clause* α *?c*)
using *sat satisfies-def relabel-def* **by** *auto*
then obtain *x* **where** *x*∈*set ?c case x of Neg n* ⇒ ¬ α *n* | *Pos n* ⇒ α *n*
using *satisfies-clause-def* **by** *auto*
then obtain *j* **where** *j*: *j* < *length ?c case* (*?c* ! *j*) *of Neg n* ⇒ ¬ α *n* | *Pos n* ⇒ α *n*
by (*metis in-set-conv-nth*)
have *case c* ! *j of Neg n* ⇒ ¬ (*remap* σ α) *n* | *Pos n* ⇒ (*remap* σ α) *n*
proof (*cases c* ! *j*)
case (*Neg n*)
then have *1*: *?c* ! *j* = *Neg* (σ ! *n*)
using *j*(*1*) **by** *simp*
have *n* ∈ *variables* φ
using *Neg j*(*1*) *nth-mem that variables-def* **by** *force*
then have *n* < *length* σ
using *assms* **by** *auto*
then show *?thesis*
using *Neg 1 j*(*2*) *remap-def* **by** *auto*
next
case (*Pos n*)

```
        then have 1: ?c ! j = Pos (σ ! n)
          using j(1) by simp
        have n ∈ variables φ
          using Pos j(1) nth-mem that variables-def by force
        then have n < length σ
          using assms by auto
        then show ?thesis
          using Pos 1 j(2) remap-def by auto
      qed
      then show ?thesis
        using satisfies-clause-def j by auto
    qed
    then show remap σ α ⊨ φ
      using satisfies-def by simp
  next
    assume sat: remap σ α ⊨ φ
    have satisfies-clause α c if c ∈ set (relabel σ φ) for c
    proof −
      let ?phi = relabel σ φ
      let ?beta = remap σ α
      obtain i where i: i < length ?phi ?phi ! i = c
        by (meson ‹c ∈ set ?phi› in-set-conv-nth)
      then have satisfies-clause ?beta (φ ! i)
          (is satisfies-clause - ?c)
        using sat satisfies-def relabel-def by simp
      then obtain x where x∈set ?c case x of Neg n ⇒ ¬ ?beta n | Pos n ⇒ ?beta n
        using satisfies-clause-def by auto
      then obtain j where j: j < length ?c case (?c ! j) of Neg n ⇒ ¬ ?beta n | Pos n ⇒ ?beta n
        by (metis in-set-conv-nth)
      then have ren: c ! j = rename σ (?c ! j)
        using i relabel-def by auto
      have case c ! j of Neg n ⇒ ¬ α n | Pos n ⇒ α n
      proof (cases ?c ! j)
        case (Neg n)
        then have ∗: c ! j = Neg (σ ! n)
          by (simp add: ren)
        have n ∈ variables φ
          using Neg i j variables-def that length-map mem-Collect-eq nth-mem relabel-def by force
        then have n < length σ
          using assms by auto
        moreover have ¬ (remap σ α) n
          using j(2) Neg by simp
        ultimately have ¬ α (σ ! n)
          using remap-def by simp
        then show ?thesis
          by (simp add: ∗)
      next
        case (Pos n)
        then have ∗: c ! j = Pos (σ ! n)
          by (simp add: ren)
        have n ∈ variables φ
          using Pos i j variables-def that length-map mem-Collect-eq nth-mem relabel-def by force
        then have n < length σ
          using assms by auto
        moreover have (remap σ α) n
          using j(2) Pos by simp
        ultimately have α (σ ! n)
          using remap-def by simp
        then show ?thesis
          by (simp add: ∗)
      qed
      moreover have length c = length (φ ! i)
        using relabel-def i by auto
```

ultimately show *?thesis*
using *satisfies-clause-def j* **by** *auto*
qed
then show $\alpha \models$ *relabel* σ φ
by (*simp add*: *satisfies-def*)
qed

4.1.3 Representing CNF formulas as strings

By identifying negated literals with even numbers and positive literals with odd numbers, we can identify literals with natural numbers. This yields a straightforward representation of a clause as a list of numbers and of a CNF formula as a list of lists of numbers. Such a list can, in turn, be represented as a symbol sequence over a quaternary alphabet as described in Section 2.9, which ultimately can be encoded over a binary alphabet (see Section 2.10). This is essentially how we represent CNF formulas as strings.

We have to introduce a bunch of functions for mapping between these representations.

fun *literal-n* :: *literal* ⇒ *nat* **where**
literal-n (*Neg i*) = *2* ∗ *i* |
literal-n (*Pos i*) = *Suc* (*2* ∗ *i*)

definition *n-literal* :: *nat* ⇒ *literal* **where**
n-literal n ≡ *if even n then Neg* (*n div 2*) *else Pos* (*n div 2*)

lemma *n-literal-n*: *n-literal* (*literal-n x*) = *x*
using *n-literal-def* **by** (*cases x*) *simp-all*

lemma *literal-n-literal*: *literal-n* (*n-literal n*) = *n*
using *n-literal-def* **by** *simp*

definition *clause-n* :: *clause* ⇒ *nat list* **where**
clause-n cl ≡ *map literal-n cl*

definition *n-clause* :: *nat list* ⇒ *clause* **where**
n-clause ns ≡ *map n-literal ns*

lemma *n-clause-n*: *n-clause* (*clause-n cl*) = *cl*
using *n-clause-def clause-n-def n-literal-n* **by** (*simp add*: *map-idI*)

lemma *clause-n-clause*: *clause-n* (*n-clause n*) = *n*
using *n-clause-def clause-n-def literal-n-literal* **by** (*simp add*: *map-idI*)

definition *formula-n* :: *formula* ⇒ *nat list list* **where**
formula-n φ ≡ *map clause-n* φ

definition *n-formula* :: *nat list list* ⇒ *formula* **where**
n-formula nss ≡ *map n-clause nss*

lemma *n-formula-n*: *n-formula* (*formula-n* φ) = φ
using *n-formula-def formula-n-def n-clause-n* **by** (*simp add*: *map-idI*)

lemma *formula-n-formula*: *formula-n* (*n-formula nss*) = *nss*
using *n-formula-def formula-n-def clause-n-clause* **by** (*simp add*: *map-idI*)

definition *formula-zs* :: *formula* ⇒ *symbol list* **where**
formula-zs φ ≡ *numlistlist* (*formula-n* φ)

The mapping between formulas and well-formed symbol sequences for lists of lists of numbers is bijective.

lemma *formula-n-inj*: *formula-n* φ_1 = *formula-n* $\varphi_2 \Longrightarrow \varphi_1 = \varphi_2$
using *n-formula-n* **by** *metis*

definition *zs-formula* :: *symbol list* ⇒ *formula* **where**
zs-formula zs ≡ *THE* φ. *formula-zs* φ = *zs*

lemma *zs-formula*:
  assumes *numlistlist-wf zs*
  shows $\exists!\varphi$. *formula-zs* φ = *zs*
proof −
  obtain *nss* **where** *nss*: *numlistlist nss* = *zs*
    using *assms numlistlist-wf-def* **by** *auto*
  let *?phi* = *n-formula nss*
  have ∗: *formula-n ?phi* = *nss*
    using *nss formula-n-formula* **by** *simp*
  then have *formula-zs ?phi* = *zs*
    using *nss formula-zs-def* **by** *simp*
  then have $\exists\varphi$. *formula-zs* φ = *zs*
    by *auto*
  moreover have φ = *?phi* **if** *formula-zs* φ = *zs* **for** φ
  proof −
    have *numlistlist* (*formula-n* φ) = *zs*
      using *that formula-zs-def* **by** *simp*
    then have *nss* = *formula-n* φ
      using *nss numlistlist-inj* **by** *simp*
    then show *?thesis*
      using *formula-n-inj* ∗ **by** *simp*
  qed
  ultimately show *?thesis*
    by *auto*
qed

lemma *zs-formula-zs*: *zs-formula* (*formula-zs* φ) = φ
  by (*simp add*: *formula-n-inj formula-zs-def numlistlist-inj the-equality zs-formula-def*)

lemma *formula-zs-formula*:
  assumes *numlistlist-wf zs*
  shows *formula-zs* (*zs-formula zs*) = *zs*
  using *assms zs-formula zs-formula-zs* **by** *metis*

There will of course be Turing machines that perform computations on formulas. In order to bound their running time, we need bounds for the length of the symbol representation of formulas.

lemma *nlength-literal-n-Pos*: *nlength* (*literal-n* (*Pos n*)) ≤ *Suc* (*nlength n*)
  using *nlength-times2plus1* **by** *simp*

lemma *nlength-literal-n-Neg*: *nlength* (*literal-n* (*Neg n*)) ≤ *Suc* (*nlength n*)
  using *nlength-times2* **by** *simp*

lemma *nlllength-formula-n*:
  fixes *V* :: *nat* **and** φ :: *formula*
  assumes $\bigwedge v.\ v \in$ *variables* $\varphi \Longrightarrow v \le V$
  shows *nlllength* (*formula-n* φ) ≤ *fsize* φ ∗ *Suc* (*Suc* (*nlength V*)) + *length* φ
  using *assms*
proof (*induction* φ)
  case *Nil*
  then show *?case*
    using *formula-n-def* **by** *simp*
next
  case (*Cons cl* φ)
  then have *0*: $\bigwedge v.\ v \in$ *variables* $\varphi \Longrightarrow v \le V$
    using *variables-def* **by** *simp*
  have *1*: $n \le V$ **if** *Pos n* ∈ *set cl* **for** *n*
    using *that variables-def* **by** (*simp add*: *Cons.prems*)
  have *2*: $n \le V$ **if** *Neg n* ∈ *set cl* **for** *n*
    using *that variables-def* **by** (*simp add*: *Cons.prems*)
  have *3*: *nlength* (*literal-n v*) ≤ *Suc* (*nlength V*) **if** *v* ∈ *set cl* **for** *v*
  proof (*cases v*)
    case (*Neg n*)

then have *nlength* (*literal-n v*) $\leq$ *Suc* (*nlength n*)
using *nlength-literal-n-Neg* **by** *blast*
moreover have $n \leq V$
using *Neg that 2* **by** *simp*
ultimately show *?thesis*
using *nlength-mono* **by** *fastforce*
next
case (*Pos n*)
then have *nlength* (*literal-n v*) $\leq$ *Suc* (*nlength n*)
using *nlength-literal-n-Pos* **by** *blast*
moreover have $n \leq V$
using *Pos that 1* **by** *simp*
ultimately show *?thesis*
using *nlength-mono* **by** *fastforce*
qed

have *nllength* (*clause-n cl*) = *length* (*numlist* (*map literal-n cl*))
using *clause-n-def nllength-def* **by** *simp*
have *nllength* (*clause-n cl*) = ($\sum$ *n*←(*map literal-n cl*). *Suc* (*nlength n*))
using *clause-n-def nllength* **by** *simp*
also have ... = ($\sum$ *v*←*cl*. *Suc* (*nlength* (*literal-n v*)))
proof −
have *map* (λ*n*. *Suc* (*nlength n*)) (*map literal-n cl*) = *map* (λ*v*. *Suc* (*nlength* (*literal-n v*))) *cl*
by *simp*
then show *?thesis*
by *metis*
qed
also have ... $\leq$ ($\sum$ *v*←*cl*. *Suc* (*Suc* (*nlength V*)))
using *sum-list-mono*[*of cl* λ*v*. *Suc* (*nlength* (*literal-n v*)) λ*v*. *Suc* (*Suc* (*nlength V*))] *3*
by *simp*
also have ... = *Suc* (*Suc* (*nlength V*)) ∗ *length cl*
using *sum-list-const* **by** *blast*
finally have *4*: *nllength* (*clause-n cl*) $\leq$ *Suc* (*Suc* (*nlength V*)) ∗ *length cl* **.**

have *concat* (*map* (λ*ns*. *numlist ns* @ [*5*]) (*map clause-n* (*cl* # φ))) =
(*numlist* (*clause-n cl*) @ [*5*]) @ *concat* (*map* (λ*ns*. *numlist ns* @ [*5*]) (*map clause-n* φ))
by *simp*
then have *length* (*concat* (*map* (λ*ns*. *numlist ns* @ [*5*]) (*map clause-n* (*cl* # φ)))) =
length ((*numlist* (*clause-n cl*) @ [*5*]) @ *concat* (*map* (λ*ns*. *numlist ns* @ [*5*]) (*map clause-n* φ)))
by *simp*
then have *nlllength* (*formula-n* (*cl* # φ)) =
length ((*numlist* (*clause-n cl*) @ [*5*]) @ *concat* (*map* (λ*ns*. *numlist ns* @ [*5*]) (*map clause-n* φ)))
using *formula-n-def numlistlist-def nlllength-def* **by** *simp*
also have ... = *length* (*numlist* (*clause-n cl*) @ [*5*]) + *length* (*concat* (*map* (λ*ns*. *numlist ns* @ [*5*]) (*map clause-n* φ)))
by *simp*
also have ... = *length* (*numlist* (*clause-n cl*) @ [*5*]) + *nlllength* (*formula-n* φ)
using *formula-n-def numlistlist-def nlllength-def* **by** *simp*
also have ... = *Suc* (*nllength* (*clause-n cl*)) + *nlllength* (*formula-n* φ)
using *nllength-def* **by** *simp*
also have ... $\leq$ *Suc* (*Suc* (*Suc* (*nlength V*)) ∗ *length cl*) + *nlllength* (*formula-n* φ)
using *4* **by** *simp*
also have ... $\leq$ *Suc* (*Suc* (*Suc* (*nlength V*)) ∗ *length cl*) + *fsize* φ ∗ *Suc* (*Suc* (*nlength V*)) + *length* φ
using *Cons 0* **by** *simp*
also have ... = *fsize* (*cl* # φ) ∗ *Suc* (*Suc* (*nlength V*)) + *length* (*cl* # φ)
by (*simp add*: *add-mult-distrib2 mult.commute fsize-def*)
finally show *?case*
by *simp*
qed

Since `SAT` is supposed to be a set of strings rather than symbol sequences, we need to map symbol sequences representing formulas to strings:

abbreviation *formula-to-string* :: *formula* ⇒ *string* **where**

formula-to-string $\varphi \equiv$ *symbols-to-string* (*binencode* (*numlistlist* (*formula-n* φ)))

lemma *formula-to-string-inj*:
assumes *formula-to-string* φ_1 = *formula-to-string* φ_2
shows $\varphi_1 = \varphi_2$
proof −
let *?xs1* = *binencode* (*numlistlist* (*formula-n* φ_1))
let *?xs2* = *binencode* (*numlistlist* (*formula-n* φ_2))
have *bin1*: *binencodable* (*numlistlist* (*formula-n* φ_1))
by (*simp add*: *Suc-le-eq numeral-2-eq-2 proper-symbols-numlistlist symbols-lt-numlistlist*)
then have *bit-symbols ?xs1*
using *bit-symbols-binencode* **by** *simp*
then have *1*: *string-to-symbols* (*symbols-to-string ?xs1*) = *?xs1*
using *bit-symbols-to-symbols* **by** *simp*

have *bin2*: *binencodable* (*numlistlist* (*formula-n* φ_2))
by (*simp add*: *Suc-le-eq numeral-2-eq-2 proper-symbols-numlistlist symbols-lt-numlistlist*)
then have *bit-symbols ?xs2*
using *bit-symbols-binencode* **by** *simp*
then have *string-to-symbols* (*symbols-to-string ?xs2*) = *?xs2*
using *bit-symbols-to-symbols* **by** *simp*
then have *?xs1* = *?xs2*
using *1 assms* **by** *simp*
then have *numlistlist* (*formula-n* φ_1) = *numlistlist* (*formula-n* φ_2)
using *binencode-inj bin1 bin2* **by** *simp*
then have *formula-n* φ_1 = *formula-n* φ_2
using *numlistlist-inj* **by** *simp*
then show *?thesis*
using *formula-n-inj* **by** *simp*
qed

While *formula-to-string* maps every CNF formula to a string, not every string represents a CNF formula. We could just ignore such invalid strings and define SAT to only contain well-formed strings. But this would implicitly place these invalid strings in the complement of SAT. While this does not cause us any issues here, it would if we were to introduce co-$\mathcal{NP}$ and wanted to show that $\overline{\texttt{SAT}}$ is in co-$\mathcal{NP}$, as we would then have to deal with the invalid strings. So it feels a little like cheating to ignore the invalid strings like this.

Arora and Barak [2, p. 45 footnote 3] recommend mapping invalid strings to "some fixed formula". A natural candidate for this fixed formula is the empty CNF, since an invalid string in a sense represents nothing, and the empty CNF formula is represented by the empty string. Since the empty CNF formula is satisfiable this implies that all invalid strings become elements of SAT.

We end up with the following definition of the protagonist of this article:

definition *SAT* :: *language* **where**
SAT $\equiv$ {*formula-to-string* φ | φ. *satisfiable* φ} $\cup$ {*x*. ¬ ($\exists\varphi$. *x* = *formula-to-string* φ)}

4.2 SAT is in $\mathcal{NP}$

In order to show that SAT is in $\mathcal{NP}$, we will construct a polynomial-time Turing machine M and specify a polynomial function p such that for every x, $x \in \texttt{SAT}$ iff. there is a $u \in \{\mathbb{O}, \mathbb{I}\}^{p(|x|)}$ such that M outputs **1** on $\langle x; u\rangle$.

The idea is straightforward: Let φ be the formula represented by the string x. Interpret the string u as a list of variables and interpret this as the assignment that assigns True to only the variables in the list. Then check if the assignment satisfies the formula. This check is "obviously" possible in polynomial time because M simply has to iterate over all clauses and check if at least one literal per clause is true under the assignment. Checking if a literal is true is simply a matter of checking whether the literal's variable is in the list u. If the assignment satisfies φ, output **1**, otherwise the empty symbol sequence.

If φ is unsatisfiable then no assignment, hence no u no matter the length will make M output **1**. On the other hand, if φ is satisfiable then there will be a satisfying assignment where a subset of the variables in φ are assigned True. Hence there will be a list of variables of at most roughly the length of φ. So setting the polynomial p to something like $p(n) = n$ should suffice.

In fact, as we shall see, $p(n) = n$ suffices. This is so because in our representation, the string x, being a list of lists, has slightly more overhead per number than the plain list u has. Therefore listing all variables in φ is guaranteed to need fewer symbols than x has.

There are several technical details to work out. First of all, the input to M need not be a well-formed pair. And if it is, the pair $\langle x, u\rangle$ has to be decoded into separate components x and u. These have to be decoded again from the binary to the quaternary alphabet, which is only possible if both x and u comprise only bit symbols (**01**). Then M needs to check if the decoded x and u are valid symbol sequences for lists (of lists) of numbers. In the case of u this is particularly finicky because the definition of $\mathcal{NP}$ requires us to provide a string u of exactly the length $p(|x|)$ and so we have to pad u with extra symbols, which have to be stripped again before the validation can take place.

In the first subsection we describe what the verifier TM has to do in terms of symbol sequences. In the subsections after that we devise a Turing machine that implements this behavior.

4.2.1 Verifying `SAT` instances

Our verifier Turing machine for `SAT` will implement the following function; that is, on input *zs* it will output *verify-sat zs*. It performs a number of decodings and well-formedness checks and outputs either **1** or the empty symbol sequence.

definition *verify-sat* :: *symbol list* ⇒ *symbol list* **where**
  verify-sat zs ≡
    let
      ys = *bindecode zs*;
      xs = *bindecode* (*first ys*);
      vs = *rstrip* ♯ (*bindecode* (*second ys*))
    in
      if even (*length* (*first ys*)) ∧ *bit-symbols* (*first ys*) ∧ *numlistlist-wf xs*
      then if bit-symbols (*second ys*) ∧ *numlist-wf vs*
            then if (λ*v*. *v* ∈ *set* (*zs-numlist vs*)) ⊨ *zs-formula xs then* [**1**] *else* []
            else []
      else [**1**]

Next we show that *verify-sat* behaves as is required of a verifier TM for `SAT`. Its polynomial running time is the subject of the next subsection.

First we consider the case that *zs* encodes a pair $\langle x, u\rangle$ of strings where x does not represent a CNF formula. Such an x is in `SAT`, hence the verifier TM is supposed to output **1**.

lemma *ex-phi-x*:
  assumes *xs* = *string-to-symbols x*
  assumes *even* (*length xs*) **and** *numlistlist-wf* (*bindecode xs*)
  shows ∃ φ. *x* = *formula-to-string* φ
proof −
  obtain *nss* **where** *numlistlist nss* = *bindecode xs*
    using *assms*(*3*) *numlistlist-wf-def* **by** *auto*
  moreover have *binencode* (*bindecode xs*) = *xs*
    using *assms*(*1*,*2*) *binencode-decode* **by** *simp*
  ultimately have *binencode* (*numlistlist nss*) = *xs*
    by *simp*
  then have *binencode* (*numlistlist* (*formula-n* (*n-formula nss*))) = *xs*
    using *formula-n-formula* **by** *simp*
  then have *formula-to-string* (*n-formula nss*) = *symbols-to-string xs*
    by *simp*
  then show *?thesis*
    using *assms*(*1*) *symbols-to-string-to-symbols* **by** *auto*
qed

lemma *verify-sat-not-wf-phi*:
  assumes *zs* = ⟨*x*; *u*⟩ **and** ¬ (∃ φ. *x* = *formula-to-string* φ)
  shows *verify-sat zs* = [**1**]
proof −
  define *ys* **where** *ys* = *bindecode zs*
  then have *first-ys*: *first ys* = *string-to-symbols x*

using *first-pair assms*(*1*) **by** *simp*
then have *2*: *bit-symbols* (*first ys*)
using *assms*(*1*) *bit-symbols-first ys-def* **by** *presburger*

define *xs* **where** *xs* = *bindecode* (*first ys*)
then have ¬ (*even* (*length* (*first ys*)) ∧ *bit-symbols* (*first ys*) ∧ *numlistlist-wf xs*)
using *first-ys ex-phi-x assms*(*2*) **by** *auto*
then show *verify-sat zs* = [**1**]
unfolding *verify-sat-def Let-def* **using** *ys-def xs-def* **by** *simp*
qed

The next case is that *zs* represents a pair $\langle x, u\rangle$ where x represents an unsatisfiable CNF formula. This x is thus not in `SAT` and the verifier TM must output something different from **1**, such as the empty string, regardless of u.

lemma *verify-sat-not-sat*:
fixes φ :: *formula*
assumes *zs* = ⟨*formula-to-string* φ; *u*⟩ **and** ¬ *satisfiable* φ
shows *verify-sat zs* = []
proof −
have *bs-phi*: *bit-symbols* (*binencode* (*formula-zs* φ))
using *bit-symbols-binencode formula-zs-def proper-symbols-numlistlist symbols-lt-numlistlist*
by (*metis Suc-le-eq dual-order.refl numeral-2-eq-2*)

define *ys* **where** *ys* = *bindecode zs*
then have *first ys* = *string-to-symbols* (*formula-to-string* φ)
using *first-pair assms*(*1*) **by** *simp*
then have *first-ys*: *first ys* = *binencode* (*formula-zs* φ)
using *bit-symbols-to-symbols bs-phi formula-zs-def* **by** *simp*
then have *2*: *bit-symbols* (*first ys*)
using *assms*(*1*) *bit-symbols-first ys-def* **by** *presburger*
have *22*: *even* (*length* (*first ys*))
using *first-ys* **by** *simp*

define *xs* **where** *xs* = *bindecode* (*first ys*)
define *vs* **where** *vs* = *rstrip 5* (*bindecode* (*second ys*))

have *wf-xs*: *numlistlist-wf xs*
using *xs-def first-ys bindecode-encode formula-zs-def numlistlist-wf-def numlistlist-wf-has2'*
by (*metis le-simps*(*3*) *numerals*(*2*))

have φ: *zs-formula xs* = φ
using *xs-def first-ys 2 binencode-decode formula-to-string-inj formula-zs-def formula-zs-formula wf-xs*
by *auto*

have *verify-sat zs* =
(*if bit-symbols* (*second ys*) ∧ *numlist-wf vs*
then if (λ*v*. *v* ∈ *set* (*zs-numlist vs*)) ⊨ φ *then* [*3*] *else* []
else [])
unfolding *verify-sat-def Let-def* **using** *ys-def xs-def vs-def 2 22 wf-xs* φ **by** *simp*
then show *verify-sat zs* = []
using *assms*(*2*) *satisfiable-def* **by** *simp*
qed

Next we consider the case in which *zs* represents a pair $\langle x, u\rangle$ where x represents a satisfiable CNF formula and u a list of numbers padded at the right with ♯ symbols. This u thus represents a variable assignment, namely the one assigning True to exactly the variables in the list. The verifier TM is to output **1** iff. this assignment satisfies the CNF formula represented by x.

First we show that stripping away ♯ symbols does not damage a symbol sequence representing a list of numbers.

lemma *rstrip-numlist-append*: *rstrip* ♯ (*numlist vars* @ *replicate pad* ♯) = *numlist vars*
(**is** *rstrip* ♯ *?zs* = *?ys*)
proof −

have $(LEAST\ i.\ i \leq length\ ?zs \wedge set\ (drop\ i\ ?zs) \subseteq \{\sharp\}) = length\ ?ys$
proof (*rule Least-equality*)
show $length\ ?ys \leq length\ ?zs \wedge set\ (drop\ (length\ ?ys)\ ?zs) \subseteq \{\sharp\}$
by *auto*
show $length\ ?ys \leq m$ **if** $m \leq length\ ?zs \wedge set\ (drop\ m\ ?zs) \subseteq \{\sharp\}$ **for** m
proof (*rule ccontr*)
assume $\neg\ length\ ?ys \leq m$
then have $m < length\ ?ys$
by *simp*
then have $?ys\ !\ m \in set\ (drop\ m\ ?ys)$
by (*metis Cons-nth-drop-Suc list.set-intros*(*1*))
moreover have $set\ (drop\ m\ ?ys) \subseteq \{\sharp\}$
using *that* **by** *simp*
ultimately have $?ys\ !\ m = \sharp$
by *auto*
moreover have $?ys\ !\ m < \sharp$
using *symbols-lt-numlist* ‹$m < length\ ?ys$› **by** *simp*
ultimately show *False*
by *simp*
qed
qed
then show *?thesis*
using *rstrip-def* **by** *simp*
qed

lemma *verify-sat-wf*:
fixes φ :: *formula* **and** *pad* :: *nat*
assumes $zs = \langle formula\text{-}to\text{-}string\ \varphi;\ symbols\text{-}to\text{-}string\ (binencode\ (numlist\ vars\ @\ replicate\ pad\ \sharp))\rangle$
shows $verify\text{-}sat\ zs = (if\ (\lambda v.\ v \in set\ vars) \models \varphi\ then\ [\mathbf{1}]\ else\ [])$
proof −
have *bs-phi*: $bit\text{-}symbols\ (binencode\ (formula\text{-}zs\ \varphi))$
using *bit-symbols-binencode formula-zs-def proper-symbols-numlistlist symbols-lt-numlistlist*
by (*metis Suc-le-eq dual-order.refl numeral-2-eq-2*)

have $binencodable\ (numlist\ vars\ @\ replicate\ pad\ \sharp)$
using *proper-symbols-numlist symbols-lt-numlist binencodable-append*[*of numlist vars replicate pad* $\sharp$]
by *fastforce*
then have *bs-vars*: $bit\text{-}symbols\ (binencode\ (numlist\ vars\ @\ replicate\ pad\ \sharp))$
using *bit-symbols-binencode* **by** *simp*

define *ys* **where** $ys = bindecode\ zs$
then have $first\ ys = string\text{-}to\text{-}symbols\ (formula\text{-}to\text{-}string\ \varphi)$
using *first-pair assms*(*1*) **by** *simp*
then have *first-ys*: $first\ ys = binencode\ (formula\text{-}zs\ \varphi)$
using *bit-symbols-to-symbols bs-phi formula-zs-def* **by** *simp*
then have *bs-first*: $bit\text{-}symbols\ (first\ ys)$
using *assms*(*1*) *bit-symbols-first ys-def* **by** *presburger*

have *even*: $even\ (length\ (first\ ys))$
using *first-ys* **by** *simp*

have *second-ys*: $second\ ys = binencode\ (numlist\ vars\ @\ replicate\ pad\ \sharp)$
using *bs-vars ys-def assms*(*1*) *bit-symbols-to-symbols second-pair* **by** *simp*
then have *bs-second*: $bit\text{-}symbols\ (second\ ys)$
using *bs-vars* **by** *simp*

define *xs* **where** $xs = bindecode\ (first\ ys)$
define *vs* **where** $vs = rstrip\ \sharp\ (bindecode\ (second\ ys))$

then have $vs = rstrip\ \sharp\ (numlist\ vars\ @\ replicate\ pad\ \sharp)$
using *second-ys* ‹$binencodable\ (numlist\ vars\ @\ replicate\ pad\ \sharp)$› *bindecode-encode* **by** *simp*
then have *vs*: $vs = numlist\ vars$
using *rstrip-numlist-append* **by** *simp*

have *wf-xs*: *numlistlist-wf xs*
using *xs-def first-ys bindecode-encode formula-zs-def numlistlist-wf-def numlistlist-wf-has2'*
by (*metis le-simps*(*3*) *numerals*(*2*))

have *verify-sat zs* =
(*if even* (*length* (*first ys*)) ∧ *bit-symbols* (*first ys*) ∧ *numlistlist-wf xs*
then if bit-symbols (*second ys*) ∧ *numlist-wf vs*
then if (λ*v*. *v* ∈ *set* (*zs-numlist vs*)) ⊨ *zs-formula xs then* [**1**] *else* []
else []
else [*3*])
unfolding *verify-sat-def Let-def* **using** *bs-second ys-def xs-def vs-def* **by** *simp*
then have ∗: *verify-sat zs* =
(*if bit-symbols* (*second ys*) ∧ *numlist-wf vs*
then if (λ*v*. *v* ∈ *set* (*zs-numlist vs*)) ⊨ *zs-formula xs then* [**1**] *else* []
else [])
unfolding *verify-sat-def Let-def* **using** *even bs-first wf-xs* **by** *simp*

have *xs* = *formula-zs* φ
using *xs-def bindecode-encode formula-zs-def first-ys proper-symbols-numlistlist symbols-lt-numlistlist*
by (*simp add*: *Suc-leI numerals*(*2*))
then have φ: φ = *zs-formula xs*
by (*simp add*: *zs-formula-zs*)

have *vars*: *vars* = *zs-numlist vs*
using *vs numlist-wf-def numlist-zs-numlist zs-numlist-ex1* **by** *blast*
then have *wf-vs*: *numlist-wf vs*
using *numlist-wf-def vs* **by** *auto*

have *verify-sat zs* = (*if* (λ*v*. *v* ∈ *set* (*zs-numlist vs*)) ⊨ *zs-formula xs then* [**1**] *else* [])
using ∗ *bs-second wf-xs wf-vs* **by** *simp*
then show *?thesis*
using φ *vars* **by** *simp*
qed

Finally we show that for every string x representing a satisfiable CNF formula there is a list of numbers representing a satisfying assignment and represented by a string of length at most $|x|$. That means there is always a string of exactly the length of x consisting of a satisfying assignment plus some padding symbols.

lemma *nllength-remove1*:
assumes *n* ∈ *set ns*
shows *nllength* (*n* # *remove1 n ns*) = *nllength ns*
using *assms nllength sum-list-map-remove1*[*of n ns* λ*n*. *Suc* (*nlength n*)] **by** *simp*

lemma *nllength-distinct-le*:
assumes *distinct ns*
and *set ns* ⊆ *set ms*
shows *nllength ns* ≤ *nllength ms*
using *assms*
proof (*induction ms arbitrary*: *ns*)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons m ms*)
show *?case*
proof (*cases m* ∈ *set ns*)
case *True*
let *?ns* = *remove1 m ns*
have *set ?ns* ⊆ *set ms*
using *Cons* **by** *auto*
moreover have *distinct ?ns*
using *Cons* **by** *simp*

ultimately have ∗: *nllength ?ns* ≤ *nllength ms*
using *Cons* **by** *simp*

have *nllength ns* = *nllength* (*m* # *?ns*)
using *True nllength-remove1* **by** *simp*
also have ... = *Suc* (*nlength m*) + *nllength ?ns*
using *nllength* **by** *simp*
also have ... ≤ *Suc* (*nlength m*) + *nllength ms*
using ∗ **by** *simp*
also have ... = *nllength* (*m* # *ms*)
using *nllength* **by** *simp*
finally show *?thesis* **.**
next
case *False*
then have *set ns* ⊆ *set ms*
using *Cons* **by** *auto*
moreover have *distinct ns*
using *Cons* **by** *simp*
ultimately have *nllength ns* ≤ *nllength ms*
using *Cons* **by** *simp*
then show *?thesis*
using *nllength* **by** *simp*
qed
qed

lemma *nlllength-nllength-concat*: *nlllength nss* = *nllength* (*concat nss*) + *length nss*
using *nlllength nllength* **by** (*induction nss*) *simp-all*

fun *variable* :: *literal* ⇒ *nat* **where**
variable (*Neg i*) = *i* |
variable (*Pos i*) = *i*

lemma *sum-list-eq*: *ns* = *ms* ⟹ *sum-list ns* = *sum-list ms*
by *simp*

lemma *nllength-clause-le*: *nllength* (*clause-n cl*) ≥ *nllength* (*map variable cl*)
proof −
have *variable x* ≤ *literal-n x* **for** *x*
by (*cases x*) *simp-all*
then have ∗: *Suc* (*nlength* (*variable x*)) ≤ *Suc* (*nlength* (*literal-n x*)) **for** *x*
using *nlength-mono* **by** *simp*

let *?ns* = *map literal-n cl*
have *nllength* (*clause-n cl*) = *nllength ?ns*
using *clause-n-def* **by** *presburger*
also have ... = ($\sum$*n*←*?ns*. *Suc* (*nlength n*))
using *nllength* **by** *simp*
also have ... = ($\sum$*x*←*cl*. *Suc* (*nlength* (*literal-n x*)))
by (*smt* (*verit*, *del-insts*) *length-map nth-equalityI nth-map*)
also have ... ≥ ($\sum$*x*←*cl*. *Suc* (*nlength* (*variable x*)))
using ∗ **by** (*simp add*: *sum-list-mono*)
finally have ($\sum$*x*←*cl*. *Suc* (*nlength* (*variable x*))) ≤ *nllength* (*clause-n cl*)
by *simp*
moreover have ($\sum$*x*←*cl*. *Suc* (*nlength* (*variable x*))) = *nllength* (*map variable cl*)
proof −
have *1*: *map* (λ*x*. *Suc* (*nlength* (*variable x*))) *cl* = *map* (λ*n*. *Suc* (*nlength n*)) (*map variable cl*)
by *simp*
then have ($\sum$*x*←*cl*. *Suc* (*nlength* (*variable x*))) = ($\sum$*n*←(*map variable cl*). *Suc* (*nlength n*))
using *sum-list-eq*[*OF 1*] **by** *simp*
then show *?thesis*
using *nllength* **by** *simp*
qed
ultimately show *?thesis*

```
    by simp
qed

lemma nlllength-formula-ge: nlllength (formula-n φ) ≥ nlllength (map (map variable) φ)
proof (induction φ)
  case Nil
  then show ?case
    by simp
next
  case (Cons cl φ)
  have nlllength (map (map variable) (cl # φ)) =
      nlllength (map (map variable) [cl]) + nlllength (map (map variable) φ)
    using nlllength by simp
  also have ... = Suc (nllength (map variable cl)) + nlllength (map (map variable) φ)
    using nlllength by simp
  also have ... ≤ Suc (nllength (map variable cl)) + nlllength (formula-n φ)
    using Cons by simp
  also have ... ≤ Suc (nllength (clause-n cl)) + nlllength (formula-n φ)
    using nllength-clause-le by simp
  also have ... = nlllength (formula-n (cl # φ))
    using nlllength by (simp add: formula-n-def)
  finally show ?case .
qed

lemma variables-shorter-formula:
  fixes φ :: formula and vars :: nat list
  assumes set vars ⊆ variables φ and distinct vars
  shows nllength vars ≤ nlllength (formula-n φ)
proof -
  let ?nss = map (map variable) φ
  have nllength (concat ?nss) ≤ nlllength ?nss
    using nlllength-nllength-concat by simp
  then have *: nllength (concat ?nss) ≤ nlllength (formula-n φ)
    using nlllength-formula-ge by (meson le-trans)

  have set vars ⊆ set (concat ?nss)
  proof
    fix n :: nat
    assume n ∈ set vars
    then have n ∈ variables φ
      using assms(1) by auto
    then obtain c where c: c ∈ set φ Neg n ∈ set c ∨ Pos n ∈ set c
      using variables-def by auto
    then obtain x where x: x ∈ set c variable x = n
      by auto
    then show n ∈ set (concat (map (map variable) φ))
      using c by auto
  qed
  then have nllength vars ≤ nllength (concat ?nss)
    using nllength-distinct-le assms(2) by simp
  then show ?thesis
    using * by simp
qed

lemma ex-assignment-linear-length:
  assumes satisfiable φ
  shows ∃ vars. (λv. v ∈ set vars) ⊨ φ ∧ nllength vars ≤ nlllength (formula-n φ)
proof -
  obtain α where α: α ⊨ φ
    using assms satisfiable-def by auto
  define poss where poss = {v. v ∈ variables φ ∧ α v}
  then have finite poss
    using finite-variables by simp
```

let *?beta* = $\lambda v.\ v \in$ *poss*
have *sat*: *?beta* $\models \varphi$
  unfolding *satisfies-def*
proof
  fix *c* :: *clause*
  assume $c \in$ *set* φ
  then have *satisfies-clause* α *c*
    using *satisfies-def* α **by** *simp*
  then obtain *x* **where** *x*: $x \in$ *set c satisfies-literal* α *x*
    using *satisfies-clause-def* **by** *auto*
  show *satisfies-clause ?beta c*
  proof (*cases x*)
    case (*Neg n*)
    then have $\neg\ \alpha\ n$
      using *x*(*2*) **by** *simp*
    then have $n \notin$ *poss*
      using *poss-def* **by** *simp*
    then have $\neg$ *?beta n*
      by *simp*
    then have *satisfies-literal ?beta x*
      using *Neg* **by** *simp*
    then show *?thesis*
      using *satisfies-clause-def x*(*1*) **by** *auto*
  next
    case (*Pos n*)
    then have α *n*
      using *x*(*2*) **by** *simp*
    then have $n \in$ *poss*
      using *poss-def Pos* ‹$c \in$ *set* φ› *variables-def x*(*1*) **by** *auto*
    then have *?beta n*
      by *simp*
    then have *satisfies-literal ?beta x*
      using *Pos* **by** *simp*
    then show *?thesis*
      using *satisfies-clause-def x*(*1*) **by** *auto*
  qed
qed

obtain *vars* **where** *vars*: *set vars* = *poss distinct vars*
  using ‹*finite poss*› **by** (*meson finite-distinct-list*)
moreover from *this* **have** *set vars* $\subseteq$ *variables* φ
  using *poss-def* **by** *simp*
ultimately have *nllength vars* $\leq$ *nlllength* (*formula-n* φ)
  using *variables-shorter-formula* **by** *simp*
moreover have ($\lambda v.\ v \in$ *set vars*) $\models \varphi$
  using *vars*(*1*) *sat* **by** *simp*
ultimately show *?thesis*
  by *auto*
qed

lemma *ex-witness-linear-length*:
  fixes φ :: *formula*
  assumes *satisfiable* φ
  shows $\exists$ *us*.
    bit-symbols us $\wedge$
    length us = *length* (*formula-to-string* φ) $\wedge$
    verify-sat ⟨*formula-to-string* φ; *symbols-to-string us*⟩ = [**1**]
proof −
  obtain *vars* **where** *vars*:
    ($\lambda v.\ v \in$ *set vars*) $\models \varphi$
    nllength vars $\leq$ *nlllength* (*formula-n* φ)
    using *assms ex-assignment-linear-length* **by** *auto*

define *pad* **where** *pad* = *nlllength* (*formula-n* φ) − *nllength vars*
then have *nllength vars* + *pad* = *nlllength* (*formula-n* φ)
  using *vars*(*2*) **by** *simp*
moreover define *us* **where** *us* = *numlist vars* @ *replicate pad* ♯
ultimately have *length us* = *nlllength* (*formula-n* φ)
  by (*simp add*: *nllength-def*)
then have *length* (*binencode us*) = *length* (*formula-to-string* φ) (**is** *length ?us* = -)
  by (*simp add*: *nlllength-def*)
moreover have *verify-sat* ⟨*formula-to-string* φ; *symbols-to-string ?us*⟩ = [**1**]
  using *us-def vars*(*1*) *assms verify-sat-wf* **by** *simp*
moreover have *bit-symbols ?us*
proof −
  have *binencodable* (*numlist vars*)
    using *proper-symbols-numlist symbols-lt-numlist*
    by (*metis Suc-leI lessI less-Suc-numeral numeral-2-eq-2 numeral-le-iff numeral-less-iff order-less-le-trans pred-numeral-simps*(*3*) *semiring-norm*(*74*))
  moreover have *binencodable* (*replicate pad* ♯)
    by *simp*
  ultimately have *binencodable us*
    using *us-def binencodable-append* **by** *simp*
  then show *?thesis*
    using *bit-symbols-binencode* **by** *simp*
qed
ultimately show *?thesis*
  by *blast*
qed

lemma *bit-symbols-verify-sat*: *bit-symbols* (*verify-sat zs*)
  unfolding *verify-sat-def Let-def* **by** *simp*

4.2.2 A Turing machine for verifying formulas

The core of the function *verify-sat* is the expression ($\lambda v.\ v \in$ *set* (*zs-numlist vs*)) $\models$ *zs-formula xs*, which checks if an assignment represented by a list of variable indices satisfies a CNF formula represented by a list of lists of literals. In this section we devise a Turing machine performing this check.

Recall that the numbers 0 and 1 are represented by the empty symbol sequence and the symbol sequence **1**, respectively. The Turing machines in this section are described in terms of numbers.

We start with a Turing machine that checks a clause. The TM accepts on tape j_1 a list of numbers representing an assignment α and on tape j_2 a list of numbers representing a clause. It outputs on tape j_3 the number 1 if α satisfies the clause, and otherwise 0. To do this the TM iterates over all literals in the clause and determines the underlying variable and the sign of the literal. If the literal is positive and the variable is in the list representing α or if the literal is negative and the variable is not in the list, the number 1 is written to the tape j_3. Otherwise the tape remains unchanged. We assume j_3 is initialized with 0, and so it will be 1 if and only if at least one literal is satisfied by α.

The TM requires five auxiliary tapes $j_3 + 1, \ldots, j_3 + 5$. Tape $j_3 + 1$ stores the literals one at a time, and later the variable; tape $j_3 + 2$ stores the sign of the literal; tape $j_3 + 3$ stores whether the variable is contained in α; tapes $j_3 + 4$ and $j_3 + 5$ are the auxiliary tapes of *tm-contains*.

definition *tm-sat-clause* :: *tapeidx* ⇒ *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-sat-clause j1 j2 j3 ≡
    WHILE [] ; λ*rs. rs* ! *j2* ≠ □ *DO*
      tm-nextract 4 j2 (*j3* + *1*) ;;
      tm-mod2 (*j3* + *1*) (*j3* + *2*) ;;
      tm-div2 (*j3* + *1*) ;;
      tm-contains j1 (*j3* + *1*) (*j3* + *3*) ;;
      IF λ*rs. rs* ! (*j3* + *3*) = □ ∧ *rs* ! (*j3* + *2*) = □ ∨ *rs* ! (*j3* + *3*) ≠ □ ∧ *rs* ! (*j3* + *2*) ≠ □ *THEN*
        tm-setn j3 1
      ELSE
        []
      ENDIF ;;
      tm-setn (*j3* + *1*) *0* ;;

tm-setn (*j3* + *2*) *0* ;;
tm-setn (*j3* + *3*) *0*
DONE ;;
tm-cr j2

lemma *tm-sat-clause-tm*:
assumes $k \geq 2$ **and** $G \geq 5$ **and** $j3 + 5 < k$ $0 < j1$ $j1 < k$ $j2 < k$ $j1 < j3$
shows *turing-machine k G* (*tm-sat-clause j1 j2 j3*)
using *tm-sat-clause-def tm-mod2-tm tm-div2-tm tm-nextract-tm tm-setn-tm tm-contains-tm Nil-tm tm-cr-tm assms turing-machine-loop-turing-machine turing-machine-branch-turing-machine*
by *simp*

locale *turing-machine-sat-clause* =
fixes *j1 j2 j3* :: *tapeidx*
begin

definition *tmL1* ≡ *tm-nextract 4 j2* (*j3* + *1*)
definition *tmL2* ≡ *tmL1* ;; *tm-mod2* (*j3* + *1*) (*j3* + *2*)
definition *tmL3* ≡ *tmL2* ;; *tm-div2* (*j3* + *1*)
definition *tmL4* ≡ *tmL3* ;; *tm-contains j1* (*j3* + *1*) (*j3* + *3*)
definition *tmI* ≡ *IF* $\lambda rs.\ rs\ !\ (j3 + 3) = \Box \wedge rs\ !\ (j3 + 2) = \Box \vee rs\ !\ (j3 + 3) \neq \Box \wedge rs\ !\ (j3 + 2) \neq \Box$ *THEN tm-setn j3 1 ELSE* [] *ENDIF*
definition *tmL5* ≡ *tmL4* ;; *tmI*
definition *tmL6* ≡ *tmL5* ;; *tm-setn* (*j3* + *1*) *0*
definition *tmL7* ≡ *tmL6* ;; *tm-setn* (*j3* + *2*) *0*
definition *tmL8* ≡ *tmL7* ;; *tm-setn* (*j3* + *3*) *0*
definition *tmL* ≡ *WHILE* [] ; $\lambda rs.\ rs\ !\ j2 \neq \Box$ *DO tmL8 DONE*
definition *tm2* ≡ *tmL* ;; *tm-cr j2*

lemma *tm2-eq-tm-sat-clause*: *tm2* = *tm-sat-clause j1 j2 j3*
unfolding *tm2-def tmL-def tmL8-def tmL7-def tmL6-def tmL5-def tmL4-def tmL3-def tmI-def tmL2-def tmL1-def tm-sat-clause-def*
by *simp*

context
fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *vars* :: *nat list* **and** *clause* :: *clause*
assumes *jk*: $0 < j1$ $j1 \neq j2$ $j3 + 5 < k$ $j1 < j3$ $j2 < j3$ $0 < j2$ *length tps0* = *k*
assumes *tps0*:
tps0 ! *j1* = *nltape′ vars 0*
tps0 ! *j2* = *nltape′* (*clause-n clause*) *0*
tps0 ! *j3* = ($\lfloor 0 \rfloor_N$, *1*)
tps0 ! (*j3* + *1*) = ($\lfloor 0 \rfloor_N$, *1*)
tps0 ! (*j3* + *2*) = ($\lfloor 0 \rfloor_N$, *1*)
tps0 ! (*j3* + *3*) = ($\lfloor 0 \rfloor_N$, *1*)
tps0 ! (*j3* + *4*) = ($\lfloor 0 \rfloor_N$, *1*)
tps0 ! (*j3* + *5*) = ($\lfloor 0 \rfloor_N$, *1*)
begin

abbreviation *sat-take t* ≡ *satisfies-clause* ($\lambda v.\ v \in$ *set vars*) (*take t clause*)

definition *tpsL* :: *nat* ⇒ *tape list* **where**
tpsL t ≡ *tps0*
[*j2* := *nltape′* (*clause-n clause*) *t*,
j3 := ($\lfloor$*sat-take t*$\rfloor_B$, *1*)]

lemma *tpsL0*: *tpsL 0* = *tps0*
proof −
have *nltape′* (*clause-n clause*) *0* = *tps0* ! *j2*
using *tps0*(*2*) **by** *presburger*
moreover have $\lfloor$*sat-take 0*$\rfloor_B$ = $\lfloor 0 \rfloor_N$
using *satisfies-clause-def* **by** *simp*
ultimately show *?thesis*
using *tpsL-def tps0 jk* **by** (*metis list-update-id*)

qed

definition *tpsL1* :: *nat* ⇒ *tape list* **where**
tpsL1 t ≡ *tps0*
[*j2* := *nltape'* (*clause-n clause*) (*Suc t*),
j3 := (⌊*sat-take t*⌋$_B$, *1*),
j3 + *1* := (⌊*literal-n* (*clause* ! *t*)⌋$_N$, *1*)]

lemma *tmL1* [*transforms-intros*]:
assumes *ttt* = *12* + *2* ∗ *nlength* (*clause-n clause* ! *t*) **and** *t* < *length* (*clause-n clause*)
shows *transforms tmL1* (*tpsL t*) *ttt* (*tpsL1 t*)
unfolding *tmL1-def*
proof (*tform tps*: *assms tps0 tpsL-def tpsL1-def jk*)
have *len*: *t* < *length clause*
using *assms*(*2*) *clause-n-def* **by** *simp*
show *ttt* = *12* + *2* ∗ *nlength 0* + *2* ∗ *nlength* (*clause-n clause* ! *t*)
using *assms*(*1*) **by** *simp*
have ∗: *j2* ≠ *j3*
using *jk* **by** *simp*
have ∗∗: *clause-n clause* ! *t* = *literal-n* (*clause* ! *t*)
using *len* **by** (*simp add*: *clause-n-def*)
show *tpsL1 t* = (*tpsL t*)
[*j2* := *nltape'* (*clause-n clause*) (*Suc t*),
j3 + *1* := (⌊*clause-n clause* ! *t*⌋$_N$, *1*)]
unfolding *tpsL-def tpsL1-def* **using** *list-update-swap*[*OF* ∗, *of tps0*] **by** (*simp add*: ∗∗)
qed

definition *tpsL2* :: *nat* ⇒ *tape list* **where**
tpsL2 t ≡ *tps0*
[*j2* := *nltape'* (*clause-n clause*) (*Suc t*),
j3 := (⌊*sat-take t*⌋$_B$, *1*),
j3 + *1* := (⌊*literal-n* (*clause* ! *t*)⌋$_N$, *1*),
j3 + *2* := (⌊*literal-n* (*clause* ! *t*) *mod 2*⌋$_N$, *1*)]

lemma *tmL2* [*transforms-intros*]:
assumes *ttt* = *12* + *2* ∗ *nlength* (*clause-n clause* ! *t*) + *1*
and *t* < *length* (*clause-n clause*)
shows *transforms tmL2* (*tpsL t*) *ttt* (*tpsL2 t*)
unfolding *tmL2-def* **by** (*tform tps*: *assms tps0 tpsL2-def tpsL1-def jk*)

definition *tpsL3* :: *nat* ⇒ *tape list* **where**
tpsL3 t ≡ *tps0*
[*j2* := *nltape'* (*clause-n clause*) (*Suc t*),
j3 := (⌊*sat-take t*⌋$_B$, *1*),
j3 + *1* := (⌊*literal-n* (*clause* ! *t*) *div 2*⌋$_N$, *1*),
j3 + *2* := (⌊*literal-n* (*clause* ! *t*) *mod 2*⌋$_N$, *1*)]

lemma *tmL3* [*transforms-intros*]:
assumes *ttt* = *16* + *4* ∗ *nlength* (*clause-n clause* ! *t*)
and *t* < *length* (*clause-n clause*)
shows *transforms tmL3* (*tpsL t*) *ttt* (*tpsL3 t*)
unfolding *tmL3-def*
proof (*tform tps*: *assms*(*2*) *tps0 tpsL3-def tpsL2-def jk*)
have *len*: *t* < *length clause*
using *assms*(*2*) *clause-n-def* **by** *simp*
have ∗∗: *clause-n clause* ! *t* = *literal-n* (*clause* ! *t*)
using *len* **by** (*simp add*: *clause-n-def*)
show *ttt* = *12* + *2* ∗ *nlength* (*clause-n clause* ! *t*) + *1* + (*2* ∗ *nlength* (*literal-n* (*clause* ! *t*)) + *3*)
using *assms*(*1*) ∗∗ **by** *simp*
qed

definition *tpsL4* :: *nat* ⇒ *tape list* **where**
tpsL4 t ≡ *tps0*

[$j2$:= *nltape'* (*clause-n clause*) (*Suc t*),
$j3$:= ($\lfloor$*sat-take t*$\rfloor_B$, *1*),
$j3$ + *1* := ($\lfloor$*literal-n* (*clause* ! *t*) *div 2*$\rfloor_N$, *1*),
$j3$ + *2* := ($\lfloor$*literal-n* (*clause* ! *t*) *mod 2*$\rfloor_N$, *1*),
$j3$ + *3* := ($\lfloor$*literal-n* (*clause* ! *t*) *div 2* ∈ *set vars*$\rfloor_B$, *1*)]

lemma *tmL4* [*transforms-intros*]:
assumes *ttt* = *20* + *4* ∗ *nlength* (*clause-n clause* ! *t*) + *67* ∗ (*nllength vars*)^2
and *t* < *length* (*clause-n clause*)
shows *transforms tmL4* (*tpsL t*) *ttt* (*tpsL4 t*)
unfolding *tmL4-def*
proof (*tform tps*: *assms*(*2*) *tps0 tpsL4-def tpsL3-def jk time*: *assms*(*1*))
have *tpsL3 t* ! ($j3$ + *4*) = ($\lfloor$*0*$\rfloor_N$, *1*)
using *tpsL3-def tps0 jk* **by** *simp*
then show *tpsL3 t* ! ($j3$ + *3* + *1*) = ($\lfloor$*0*$\rfloor_N$, *1*)
by (*metis ab-semigroup-add-class.add-ac*(*1*) *numeral-plus-one semiring-norm*(*2*) *semiring-norm*(*8*))
have *tpsL3 t* ! ($j3$ + *5*) = ($\lfloor$*0*$\rfloor_N$, *1*)
using *tpsL3-def tps0 jk* **by** *simp*
then show *tpsL3 t* ! ($j3$ + *3* + *2*) = ($\lfloor$*0*$\rfloor_N$, *1*)
by (*simp add*: *numeral-Bit1*)
qed

definition *tpsL5* :: *nat* ⇒ *tape list* **where**
tpsL5 t ≡ *tps0*
[$j2$:= *nltape'* (*clause-n clause*) (*Suc t*),
$j3$:= ($\lfloor$*sat-take* (*Suc t*)$\rfloor_B$, *1*),
$j3$ + *1* := ($\lfloor$*literal-n* (*clause* ! *t*) *div 2*$\rfloor_N$, *1*),
$j3$ + *2* := ($\lfloor$*literal-n* (*clause* ! *t*) *mod 2*$\rfloor_N$, *1*),
$j3$ + *3* := ($\lfloor$*literal-n* (*clause* ! *t*) *div 2* ∈ *set vars*$\rfloor_B$, *1*)]

lemma *tmI* [*transforms-intros*]:
assumes *ttt* = *16* **and** *t* < *length* (*clause-n clause*)
shows *transforms tmI* (*tpsL4 t*) *ttt* (*tpsL5 t*)
unfolding *tmI-def*
proof (*tform tps*: *jk tpsL4-def time*: *assms*(*1*))
show *10* + *2* ∗ *nlength* (*if sat-take t then 1 else 0*) + *2* ∗ *nlength 1* + *2* ≤ *ttt*
using *assms*(*1*) *nlength-0 nlength-1-simp* **by** *simp*

have *len*: *t* < *length clause*
using *assms*(*2*) **by** (*simp add*: *clause-n-def*)

let *?l* = *clause* ! *t*
have *1*: *read* (*tpsL4 t*) ! ($j3$ + *3*) = □ ⟷ *literal-n ?l div 2* ∉ *set vars*
using *tpsL4-def jk read-ncontents-eq-0*[*of tpsL4 t* $j3$ + *3*] **by** *simp*
have *2*: *read* (*tpsL4 t*) ! ($j3$ + *2*) = □ ⟷ *literal-n ?l mod 2* = *0*
using *tpsL4-def jk read-ncontents-eq-0*[*of tpsL4 t* $j3$ + *2*] **by** *simp*

let *?a* = *λv. v* ∈ *set vars*
let *?cond* = *read* (*tpsL4 t*) ! ($j3$ + *3*) = □ ∧ *read* (*tpsL4 t*) ! ($j3$ + *2*) = □ ∨
read (*tpsL4 t*) ! ($j3$ + *3*) ≠ □ ∧ *read* (*tpsL4 t*) ! ($j3$ + *2*) ≠ □
have ∗: *?cond* ⟷ *satisfies-literal ?a ?l*
proof (*cases ?l*)
case (*Neg v*)
then have *literal-n ?l div 2* = *v literal-n ?l mod 2* = *0*
by *simp-all*
moreover from *this* **have** *satisfies-literal ?a ?l* ⟷ *v* ∉ *set vars*
using *Neg* **by** *simp*
ultimately show *?thesis*
using *1 2* **by** *simp*
next
case (*Pos v*)
then have *literal-n ?l div 2* = *v literal-n ?l mod 2* = *1*
by *simp-all*

moreover from *this* **have** *satisfies-literal ?a ?l* ⟷ *v* ∈ *set vars*
using *Pos* **by** *simp*
ultimately show *?thesis*
using *1 2* **by** *simp*
qed

have ∗∗: *sat-take* (*Suc t*) ⟷ *sat-take t* ∨ *satisfies-literal ?a ?l*
using *satisfies-clause-take*[*OF len*] **by** *simp*

show *tpsL5 t* = (*tpsL4 t*)[*j3* := (⌊*1*⌋$_N$, *1*)] **if** *?cond*
proof −
have (*if sat-take* (*Suc t*) *then 1*::*nat else 0*) = *1*
using *that* ∗ ∗∗ **by** *simp*
then show *?thesis*
unfolding *tpsL5-def tpsL4-def* **using** *that* **by** (*simp add*: *list-update-swap*)
qed
show *tpsL5 t* = (*tpsL4 t*) **if** ¬ *?cond*
proof −
have *sat-take t* = *sat-take* (*Suc t*)
using ∗ ∗∗ *that* **by** *simp*
then show *?thesis*
unfolding *tpsL5-def tpsL4-def* **using** *that* **by** (*simp add*: *list-update-swap*)
qed
qed

lemma *tmL5* [*transforms-intros*]:
assumes *ttt* = *36* + *4* ∗ *nlength* (*clause-n clause* ! *t*) + *67* ∗ (*nllength vars*)^2
and *t* < *length* (*clause-n clause*)
shows *transforms tmL5* (*tpsL t*) *ttt* (*tpsL5 t*)
unfolding *tmL5-def* **by** (*tform tps*: *assms*)

definition *tpsL6* :: *nat* ⇒ *tape list* **where**
tpsL6 t ≡ *tps0*
[*j2* := *nltape′* (*clause-n clause*) (*Suc t*),
j3 := (⌊*sat-take* (*Suc t*)⌋$_B$, *1*),
j3 + *1* := (⌊*0*⌋$_N$, *1*),
j3 + *2* := (⌊*literal-n* (*clause* ! *t*) *mod 2*⌋$_N$, *1*),
j3 + *3* := (⌊*literal-n* (*clause* ! *t*) *div 2* ∈ *set vars*⌋$_B$, *1*)]

lemma *tmL6* [*transforms-intros*]:
assumes *ttt* = *46* + *4* ∗ *nlength* (*clause-n clause* ! *t*) + *67* ∗ (*nllength vars*)^2 + *2* ∗ *nlength* (*literal-n* (*clause* ! *t*) *div 2*)
and *t* < *length* (*clause-n clause*)
shows *transforms tmL6* (*tpsL t*) *ttt* (*tpsL6 t*)
unfolding *tmL6-def* **by** (*tform tps*: *assms tps0 tpsL6-def tpsL5-def jk*)

definition *tpsL7* :: *nat* ⇒ *tape list* **where**
tpsL7 t ≡ *tps0*
[*j2* := *nltape′* (*clause-n clause*) (*Suc t*),
j3 := (⌊*sat-take* (*Suc t*)⌋$_B$, *1*),
j3 + *1* := (⌊*0*⌋$_N$, *1*),
j3 + *2* := (⌊*0*⌋$_N$, *1*),
j3 + *3* := (⌊*literal-n* (*clause* ! *t*) *div 2* ∈ *set vars*⌋$_B$, *1*)]

lemma *tmL7* [*transforms-intros*]:
assumes *ttt* = *56* + *4* ∗ *nlength* (*clause-n clause* ! *t*) + *67* ∗ (*nllength vars*)^2 + *2* ∗ *nlength* (*literal-n* (*clause* ! *t*) *div 2*) +
2 ∗ *nlength* (*literal-n* (*clause* ! *t*) *mod 2*)
and *t* < *length* (*clause-n clause*)
shows *transforms tmL7* (*tpsL t*) *ttt* (*tpsL7 t*)
unfolding *tmL7-def* **by** (*tform tps*: *assms tps0 tpsL7-def tpsL6-def jk*)

definition *tpsL8* :: *nat* ⇒ *tape list* **where**

tpsL8 t ≡ *tps0*
[*j2* := *nltape'* (*clause-n clause*) (*Suc t*),
j3 := (⌊*sat-take* (*Suc t*)⌋$_B$, *1*),
j3 + *1* := (⌊*0*⌋$_N$, *1*),
j3 + *2* := (⌊*0*⌋$_N$, *1*),
j3 + *3* := (⌊*0*⌋$_N$, *1*)]

lemma *tmL8*:
assumes $ttt = 66 + 4 * nlength\ (clause\text{-}n\ clause\ !\ t) + 67 * (nllength\ vars)^2 +$
$2 * nlength\ (literal\text{-}n\ (clause\ !\ t)\ div\ 2) +$
$2 * nlength\ (literal\text{-}n\ (clause\ !\ t)\ mod\ 2) +$
$2 * nlength\ (if\ literal\text{-}n\ (clause\ !\ t)\ div\ 2 \in set\ vars\ then\ 1\ else\ 0)$
and $t < length\ (clause\text{-}n\ clause)$
shows *transforms tmL8* (*tpsL t*) *ttt* (*tpsL8 t*)
unfolding *tmL8-def* **by** (*tform tps*: *assms tps0 tpsL8-def tpsL7-def jk*)

lemma *tmL8'*:
assumes $ttt = 70 + 6 * nllength\ (clause\text{-}n\ clause) + 67 * (nllength\ vars)^2$
and $t < length\ (clause\text{-}n\ clause)$
shows *transforms tmL8* (*tpsL t*) *ttt* (*tpsL8 t*)
proof −
let $?l = literal\text{-}n\ (clause\ !\ t)$
let $?ll = clause\text{-}n\ clause\ !\ t$
let $?t = 66 + 4 * nlength\ ?ll + 67 * (nllength\ vars)^2 +$
$2 * nlength\ (?l\ div\ 2) + 2 * nlength\ (?l\ mod\ 2) + 2 * nlength\ (if\ ?l\ div\ 2 \in set\ vars\ then\ 1\ else\ 0)$
have $?t = 66 + 4 * nlength\ ?ll + 67 * (nllength\ vars)^2 +$
$2 * nlength\ (?ll\ div\ 2) + 2 * nlength\ (?ll\ mod\ 2) + 2 * nlength\ (if\ ?ll\ div\ 2 \in set\ vars\ then\ 1\ else\ 0)$
using *assms*(*2*) *clause-n-def* **by** *simp*
also have ... $\le 66 + 4 * nlength\ ?ll + 67 * (nllength\ vars)^2 +$
$2 * nlength\ ?ll + 2 * nlength\ (?ll\ mod\ 2) + 2 * nlength\ (if\ ?ll\ div\ 2 \in set\ vars\ then\ 1\ else\ 0)$
using *nlength-mono*[*of ?ll div 2 ?ll*] **by** *simp*
also have ... $= 66 + 6 * nlength\ ?ll + 67 * (nllength\ vars)^2 +$
$2 * nlength\ (?ll\ mod\ 2) + 2 * nlength\ (if\ ?ll\ div\ 2 \in set\ vars\ then\ 1\ else\ 0)$
by *simp*
also have ... $\le 66 + 6 * nlength\ ?ll + 67 * (nllength\ vars)^2 +$
$2 * nlength\ 1 + 2 * nlength\ (if\ ?ll\ div\ 2 \in set\ vars\ then\ 1\ else\ 0)$
using *nlength-mono* **by** *simp*
also have ... $\le 66 + 6 * nlength\ ?ll + 67 * (nllength\ vars)^2 + 2 * nlength\ 1 + 2 * nlength\ 1$
using *nlength-mono* **by** *simp*
also have ... $= 70 + 6 * nlength\ ?ll + 67 * (nllength\ vars)^2$
using *nlength-1-simp* **by** *simp*
also have ... $\le 70 + 6 * nllength\ (clause\text{-}n\ clause) + 67 * (nllength\ vars)^2$
using *assms*(*2*) *member-le-nllength* **by** *simp*
finally have $?t \le ttt$
using *assms*(*1*) **by** *simp*
then show *?thesis*
using *assms tmL8 transforms-monotone* **by** *blast*
qed

definition *tpsL8'* :: *nat* ⇒ *tape list* **where**
tpsL8' t ≡ *tps0*
[*j2* := *nltape'* (*clause-n clause*) (*Suc t*),
j3 := (⌊*sat-take* (*Suc t*)⌋$_B$, *1*)]

lemma *tpsL8'*: *tpsL8'* = *tpsL8*
proof −
{ **fix** *t* :: *nat*
have *tpsL8 t* = *tps0*
[*j2* := *nltape'* (*clause-n clause*) (*Suc t*),
j3 := (⌊*sat-take* (*Suc t*)⌋$_B$, *1*),
j3 + *1* := (⌊*0*⌋$_N$, *1*),
j3 + *2* := (⌊*0*⌋$_N$, *1*)]
unfolding *tpsL8-def*

```
      using tps0 list-update-id[of tps0 j3 + 3] jk
      by (simp add: list-update-swap[of - j3 + 3])
    also have ... = tps0
      [j2 := nltape′ (clause-n clause) (Suc t),
       j3 := (⌊sat-take (Suc t)⌋B, 1),
       j3 + 1 := (⌊0⌋N, 1)]
      unfolding tpsL8-def
      using tps0 list-update-id[of tps0 j3 + 2] jk
      by (simp add: list-update-swap[of - Suc (Suc j3)])
    also have ... = tps0
      [j2 := nltape′ (clause-n clause) (Suc t),
       j3 := (⌊sat-take (Suc t)⌋B, 1)]
      unfolding tpsL8-def
      using tps0 list-update-id[of tps0 j3 + 1] jk
      by (simp add: list-update-swap[of - Suc j3])
    also have ... = tpsL8′ t
      using tpsL8′-def by simp
    finally have tpsL8 t = tpsL8′ t .
  }
  then show ?thesis
    by auto
qed

lemma tmL8′′ [transforms-intros]:
  assumes ttt = 70 + 6 ∗ nllength (clause-n clause) + 67 ∗ (nllength vars)²
    and t < length (clause-n clause)
  shows transforms tmL8 (tpsL t) ttt (tpsL8′ t)
  using tmL8′ tpsL8′ assms by simp

lemma tmL [transforms-intros]:
  assumes ttt = length (clause-n clause) ∗ (72 + 6 ∗ nllength (clause-n clause) + 67 ∗ (nllength vars)²) + 1
  shows transforms tmL (tpsL 0) ttt (tpsL (length (clause-n clause)))
  unfolding tmL-def
proof (tform)
  let ?t = 70 + 6 ∗ nllength (clause-n clause) + 67 ∗ (nllength vars)²
  have tpsL8′ t = tpsL (Suc t) for t
    using tpsL8′-def tpsL-def by simp
  then show ⋀i. i < length (clause-n clause) ⟹ transforms tmL8 (tpsL i) ?t (tpsL (Suc i))
    using tmL8′′ by simp

  let ?ns = clause-n clause
  have ∗: tpsL t ! j2 = nltape′ ?ns t for t
    using tpsL-def jk by simp
  moreover have read (tpsL t) ! j2 = tpsL t :.: j2 for t
    using tapes-at-read′[of j2 tpsL t] tpsL-def jk by simp
  ultimately have read (tpsL t) ! j2 = |.| (nltape′ ?ns t) for t
    by simp
  then have read (tpsL t) ! j2 = □ ⟷ (t ≥ length ?ns) for t
    using nltape′-tape-read by simp
  then show
      ⋀i. i < length ?ns ⟹ read (tpsL i) ! j2 ≠ □
      ¬ read (tpsL (length ?ns)) ! j2 ≠ □
    using ∗ by simp-all

  show length ?ns ∗ (70 + 6 ∗ nllength ?ns + 67 ∗ (nllength vars)² + 2) + 1 ≤ ttt
    using assms by simp
qed

definition tps1 :: tape list where
  tps1 ≡ tps0
    [j2 := nltape′ (clause-n clause) (length (clause-n clause)),
     j3 := (⌊satisfies-clause (λv. v ∈ set vars) clause⌋B, 1)]
```

```
lemma tps1: tps1 = tpsL (length (clause-n clause))
proof −
  have length (clause-n clause) = length clause
    by (simp add: clause-n-def)
  then show ?thesis
    using tps1-def tpsL-def by simp
qed

lemma tm1 [transforms-intros]:
  assumes ttt = length (clause-n clause) ∗ (72 + 6 ∗ nllength (clause-n clause) + 67 ∗ (nllength vars)^2) + 1
  shows transforms tmL tps0 ttt tps1
  using tmL tpsL0 assms tps1 by simp

definition tps2 :: tape list where
  tps2 ≡ tps0
    [j2 := nltape' (clause-n clause) 0,
     j3 := (⌊satisfies-clause (λv. v ∈ set vars) clause⌋B, 1)]

lemma tm2:
  assumes ttt = length (clause-n clause) ∗ (72 + 6 ∗ nllength (clause-n clause) + 67 ∗ (nllength vars)^2) +
      nllength (clause-n clause) + 4
  shows transforms tm2 tps0 ttt tps2
  unfolding tm2-def
proof (tform tps: assms tps0 tps1-def jk)
  have ∗: tps1 ! j2 = nltape' (clause-n clause) (length (clause-n clause))
    using tps1-def jk by simp
  then show clean-tape (tps1 ! j2)
    using clean-tape-nlcontents by simp
  have neq: j3 ≠ j2
    using jk by simp
  have tps2 = tps1[j2 := nltape' (clause-n clause) 0]
    unfolding tps2-def tps1-def by (simp add: list-update-swap[OF neq])
  moreover have tps1 ! j2 |#=| 1 = nltape' (clause-n clause) 0
    using ∗ by simp
  ultimately show tps2 = tps1[j2 := tps1 ! j2 |#=| 1]
    by simp
qed

definition tps2' :: tape list where
  tps2' ≡ tps0
    [j3 := (⌊satisfies-clause (λv. v ∈ set vars) clause⌋B, 1)]

lemma tm2':
  assumes ttt = 79 ∗ (nllength (clause-n clause)) ^ 2 + 67 ∗ (nllength (clause-n clause)) ∗ nllength vars ^ 2 +
4
  shows transforms tm2 tps0 ttt tps2'
proof −
  let ?l = nllength (clause-n clause)
  let ?t = length (clause-n clause) ∗ (72 + 6 ∗ ?l + 67 ∗ (nllength vars)^2) + ?l + 4
  have ?t ≤ ?l ∗ (72 + 6 ∗ ?l + 67 ∗ (nllength vars)^2) + ?l + 4
    by (simp add: length-le-nllength)
  also have ... = ?l ∗ (73 + 6 ∗ ?l + 67 ∗ (nllength vars)^2) + 4
    by algebra
  also have ... = 73 ∗ ?l + 6 ∗ ?l ^ 2 + 67 ∗ ?l ∗ (nllength vars)^2 + 4
    by algebra
  also have ... ≤ 79 ∗ ?l ^ 2 + 67 ∗ ?l ∗ (nllength vars)^2 + 4
    using linear-le-pow by simp
  finally have ?t ≤ ttt
    using assms by simp
  moreover have tps2' = tps2
    unfolding tps2'-def tps2-def using jk tps0 by (metis tape-list-eq)
  ultimately show ?thesis
    using tps2'-def tm2 assms transforms-monotone by simp
```

qed

end

end

lemma *transforms-tm-sat-clauseI* [*transforms-intros*]:
  fixes *j1 j2 j3* :: *tapeidx*
  fixes *tps tps′* :: *tape list* **and** *ttt k* :: *nat* **and** *vars* :: *nat list* **and** *clause* :: *literal list*
  assumes *0 < j1 j1 ≠ j2 j3 + 5 < k j1 < j3 j2 < j3 0 < j2 length tps = k*
  assumes
    tps ! j1 = nltape′ vars 0
    tps ! j2 = nltape′ (clause-n clause) 0
    tps ! j3 = (⌊0⌋_N, 1)
    tps ! (j3 + 1) = (⌊0⌋N, 1)
    tps ! (j3 + 2) = (⌊0⌋N, 1)
    tps ! (j3 + 3) = (⌊0⌋N, 1)
    tps ! (j3 + 4) = (⌊0⌋N, 1)
    tps ! (j3 + 5) = (⌊0⌋N, 1)
  assumes *tps′ = tps*
    [j3 := (⌊satisfies-clause (λv. v ∈ set vars) clause⌋B, 1)]
  assumes *ttt = 79 ∗ (nllength (clause-n clause)) ^ 2 + 67 ∗ (nllength (clause-n clause)) ∗ nllength vars ^ 2 + 4*
  shows *transforms (tm-sat-clause j1 j2 j3) tps ttt tps′*
proof −
  interpret *loc*: *turing-machine-sat-clause j1 j2 j3* **.**
  show *?thesis*
    using *assms loc.tps2′-def loc.tm2′ loc.tm2-eq-tm-sat-clause* **by** *simp*
qed

The following Turing machine expects a list of lists of numbers representing a formula φ on tape j_1 and a list of numbers representing an assignment α on tape j_2. It outputs on tape j_3 the number 1 if α satisfies φ, and otherwise the number 0. To do so the TM iterates over all clauses in φ and uses *tm-sat-clause* on each of them. It requires seven auxiliary tapes: $j_3 + 1$ to store the clauses one at a time, $j_3 + 2$ to store the results of *tm-sat-clause*, whose auxiliary tapes are $j_3 + 3, \ldots, j_3 + 7$.

definition *tm-sat-formula* :: *tapeidx ⇒ tapeidx ⇒ tapeidx ⇒ machine* **where**
  tm-sat-formula j1 j2 j3 ≡
    tm-setn j3 1 ;;
    WHILE [] ; *λrs. rs ! j1 ≠ □ DO*
      tm-nextract ♯ j1 (j3 + 1) ;;
      tm-sat-clause j2 (j3 + 1) (j3 + 2) ;;
      IF λrs. rs ! (j3 + 2) = □ THEN
        tm-setn j3 0
      ELSE
        []
      ENDIF ;;
      tm-erase-cr (j3 + 1) ;;
      tm-setn (j3 + 2) 0
    DONE

lemma *tm-sat-formula-tm*:
  assumes *k ≥ 2* **and** *G ≥ 6* **and** *0 < j1 j1 ≠ j2 j3 + 7 < k j1 < j3 j2 < j3 0 < j2*
  shows *turing-machine k G (tm-sat-formula j1 j2 j3)*
  using *tm-sat-formula-def tm-sat-clause-tm tm-nextract-tm tm-setn-tm assms Nil-tm tm-erase-cr-tm*
    turing-machine-loop-turing-machine turing-machine-branch-turing-machine
  by *simp*

locale *turing-machine-sat-formula* =
  fixes *j1 j2 j3* :: *tapeidx*
begin

definition *tm1* ≡ *tm-setn j3 1*

definition *tmL1* ≡ *tm-nextract* ♯ *j1* (*j3* + *1*)
definition *tmL2* ≡ *tmL1* ;; *tm-sat-clause j2* (*j3* + *1*) (*j3* + *2*)
definition *tmI* ≡ *IF* λ*rs. rs* ! (*j3* + *2*) = □ *THEN tm-setn j3 0 ELSE* [] *ENDIF*
definition *tmL3* ≡ *tmL2* ;; *tmI*
definition *tmL4* ≡ *tmL3* ;; *tm-erase-cr* (*j3* + *1*)
definition *tmL5* ≡ *tmL4* ;; *tm-setn* (*j3* + *2*) *0*
definition *tmL* ≡ *WHILE* [] ; λ*rs. rs* ! *j1* ≠ □ *DO tmL5 DONE*

definition *tm2* ≡ *tm1* ;; *tmL*

lemma *tm2-eq-tm-sat-formula*: *tm2* = *tm-sat-formula j1 j2 j3*
  unfolding *tm2-def tm1-def tmL-def tmL5-def tmL4-def tmL3-def tmI-def tmL2-def tmL1-def tm-sat-formula-def*
  by *simp*

context
  fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *vars* :: *nat list* **and** φ :: *formula*
  assumes *jk*: *0* < *j1 j1* ≠ *j2 j3* + *7* < *k j1* < *j3 j2* < *j3 0* < *j2 length tps0* = *k*
  assumes *tps0*:
    tps0 ! *j1* = *nlltape′* (*formula-n* φ) *0*
    tps0 ! *j2* = *nltape′ vars 0*
    tps0 ! *j3* = (⌊*0*⌋$_N$, *1*)
    tps0 ! (*j3* + *1*) = (⌊[]⌋$_{NL}$, *1*)
    tps0 ! (*j3* + *2*) = (⌊*0*⌋$_N$, *1*)
    tps0 ! (*j3* + *3*) = (⌊*0*⌋$_N$, *1*)
    tps0 ! (*j3* + *4*) = (⌊*0*⌋$_N$, *1*)
    tps0 ! (*j3* + *5*) = (⌊*0*⌋$_N$, *1*)
    tps0 ! (*j3* + *6*) = (⌊*0*⌋$_N$, *1*)
    tps0 ! (*j3* + *7*) = (⌊*0*⌋$_N$, *1*)
begin

definition *tps1* ≡ *tps0*
  [*j3* := (⌊*1*⌋$_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
  assumes *ttt* = *12*
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def*
proof (*tform tps*: *tps0 tps1-def jk*)
  show *ttt* = *10* + *2* ∗ *nlength 0* + *2* ∗ *nlength 1*
    using *assms nlength-1-simp* **by** *simp*
qed

abbreviation *sat-take t* ≡ (λ*v. v* ∈ *set vars*) ⊨ *take t* φ

definition *tpsL* :: *nat* ⇒ *tape list* **where**
  tpsL t ≡ *tps0*
    [*j1* := *nlltape′* (*formula-n* φ) *t*,
     j3 := (⌊*sat-take t*⌋$_B$, *1*)]

lemma *tpsL0*: *tpsL 0* = *tps1*
proof −
  have *nlltape′* (*formula-n* φ) *0* = *tps1* ! *j1*
    using *tps0*(*1*) *tps1-def jk* **by** *simp*
  moreover have ⌊*sat-take 0*⌋$_B$ = ⌊*1*⌋$_N$
    using *satisfies-def* **by** *simp*
  ultimately show *?thesis*
    using *tpsL-def tps0 jk tps1-def* **by** (*metis list-update-id*)
qed

definition *tpsL1* :: *nat* ⇒ *tape list* **where**
  tpsL1 t ≡ *tps0*
    [*j1* := *nlltape′* (*formula-n* φ) (*Suc t*),
     j3 := (⌊*sat-take t*⌋$_B$, *1*),

$j3 + 1 := (\lfloor formula\text{-}n\ \varphi\ !\ t \rfloor_{NL},\ 1)]$

lemma *tmL1* [*transforms-intros*]:
  assumes $ttt = 12 + 2 * nllength\ (formula\text{-}n\ \varphi\ !\ t)$ **and** $t < length\ (formula\text{-}n\ \varphi)$
  shows *transforms tmL1 (tpsL t) ttt (tpsL1 t)*
  unfolding *tmL1-def*
proof (*tform tps*: *assms tps0 tpsL-def tpsL1-def jk*)
  show $ttt = 12 + 2 * nllength\ [] + 2 * nllength\ (formula\text{-}n\ \varphi\ !\ t)$
    using *assms*(*1*) **by** *simp*
  show $tpsL1\ t = (tpsL\ t)$
    $[j1 := nlltape'\ (formula\text{-}n\ \varphi)\ (Suc\ t),$
     $j3 + 1 := (\lfloor formula\text{-}n\ \varphi\ !\ t \rfloor_{NL},\ 1)]$
    using *tpsL1-def tpsL-def jk* **by** (*simp add*: *list-update-swap*)
qed

definition $tpsL2 :: nat \Rightarrow tape\ list$ **where**
  $tpsL2\ t \equiv tps0$
    $[j1 := nlltape'\ (formula\text{-}n\ \varphi)\ (Suc\ t),$
     $j3 := (\lfloor sat\text{-}take\ t \rfloor_{B},\ 1),$
     $j3 + 1 := (\lfloor formula\text{-}n\ \varphi\ !\ t \rfloor_{NL},\ 1),$
     $j3 + 2 := (\lfloor satisfies\text{-}clause\ (\lambda v.\ v \in set\ vars)\ (\varphi\ !\ t) \rfloor_{B},\ 1)]$

lemma *tmL2* [*transforms-intros*]:
  assumes $ttt = 12 + 2 * nllength\ (formula\text{-}n\ \varphi\ !\ t) +$
    $(79 * (nllength\ (formula\text{-}n\ \varphi\ !\ t))^2 +$
     $67 * nllength\ (formula\text{-}n\ \varphi\ !\ t) * (nllength\ vars)^2 + 4)$
    and $t < length\ (formula\text{-}n\ \varphi)$
  shows *transforms tmL2 (tpsL t) ttt (tpsL2 t)*
  unfolding *tmL2-def*
proof (*tform tps*: *assms tps0 tpsL-def tpsL1-def jk*)
  let $?clause = \varphi\ !\ t$
  have $*$: $formula\text{-}n\ \varphi\ !\ t = clause\text{-}n\ ?clause$
    using *assms*(*2*) *formula-n-def* **by** *simp*
  then have $(\lfloor formula\text{-}n\ \varphi\ !\ t \rfloor_{NL},\ 1) = nltape'\ (clause\text{-}n\ ?clause)\ 0$
    by *simp*
  then show $tpsL1\ t\ !\ (j3 + 1) = nltape'\ (clause\text{-}n\ ?clause)\ 0$
    using *tpsL1-def jk* **by** *simp*
  have $j3 + 2 + 1 = j3 + 3$
    by *simp*
  moreover have $tpsL1\ t\ !\ (j3 + 3) = (\lfloor 0 \rfloor_{N},\ 1)$
    using *tpsL1-def tps0 jk* **by** *simp*
  ultimately show $tpsL1\ t\ !\ (j3 + 2 + 1) = (\lfloor 0 \rfloor_{N},\ 1)$
    by *metis*
  have $j3 + 2 + 2 = j3 + 4$
    by *simp*
  moreover have $tpsL1\ t\ !\ (j3 + 4) = (\lfloor 0 \rfloor_{N},\ 1)$
    using *tpsL1-def tps0 jk* **by** *simp*
  ultimately show $tpsL1\ t\ !\ (j3 + 2 + 2) = (\lfloor 0 \rfloor_{N},\ 1)$
    by *metis*
  have $j3 + 2 + 3 = j3 + 5$
    by *simp*
  moreover have $tpsL1\ t\ !\ (j3 + 5) = (\lfloor 0 \rfloor_{N},\ 1)$
    using *tpsL1-def tps0 jk* **by** *simp*
  ultimately show $tpsL1\ t\ !\ (j3 + 2 + 3) = (\lfloor 0 \rfloor_{N},\ 1)$
    by *metis*
  have $j3 + 2 + 4 = j3 + 6$
    by *simp*
  moreover have $tpsL1\ t\ !\ (j3 + 6) = (\lfloor 0 \rfloor_{N},\ 1)$
    using *tpsL1-def tps0 jk* **by** *simp*
  ultimately show $tpsL1\ t\ !\ (j3 + 2 + 4) = (\lfloor 0 \rfloor_{N},\ 1)$
    by *metis*
  have $j3 + 2 + 5 = j3 + 7$
    by *simp*

moreover have *tpsL1 t* ! (*j3* + *7*) = (⌊*0*⌋$_N$, *1*)
  using *tpsL1-def tps0 jk* **by** *simp*
ultimately show *tpsL1 t* ! (*j3* + *2* + *5*) = (⌊*0*⌋$_N$, *1*)
  by *metis*
show *tpsL2 t* = (*tpsL1 t*)
    [*j3* + *2* := (⌊*satisfies-clause* (λ*v*. *v* ∈ *set vars*) (φ ! *t*)⌋$_B$, *1*)]
  unfolding *tpsL2-def tpsL1-def* **by** *simp*
show *ttt* = *12* + *2* ∗ *nllength* (*formula-n* φ ! *t*) +
    (*79* ∗ (*nllength* (*clause-n* (φ ! *t*)))^2 +
    67 ∗ *nllength* (*clause-n* (φ ! *t*)) ∗ (*nllength vars*)^2 + *4*)
  using *assms*(*1*) ∗ **by** *simp*
qed

definition *tpsL3* :: *nat* ⇒ *tape list* **where**
  tpsL3 t ≡ *tps0*
    [*j1* := *nlltape*′ (*formula-n* φ) (*Suc t*),
    j3 := (⌊*sat-take* (*Suc t*)⌋$_B$, *1*),
    j3 + *1* := (⌊*formula-n* φ ! *t*⌋$_{NL}$, *1*),
    j3 + *2* := (⌊*satisfies-clause* (λ*v*. *v* ∈ *set vars*) (φ ! *t*)⌋$_B$, *1*)]

lemma *tmI* [*transforms-intros*]:
  assumes *ttt* = *16* **and** *t* < *length* (*formula-n* φ)
  shows *transforms tmI* (*tpsL2 t*) *ttt* (*tpsL3 t*)
  unfolding *tmI-def*
proof (*tform tps*: *assms*(*2*) *tps0 tpsL2-def tpsL3-def jk time*: *assms*(*1*))
  show *10* + *2* ∗ *nlength* (*if sat-take t then 1 else 0*) + *2* ∗ *nlength 0* + *2* ≤ *ttt*
    using *assms*(*1*) *nlength-1-simp* **by** *simp*

  let *?a* = λ*v*. *v* ∈ *set vars*
  let *?cl* = φ ! *t*
  have ∗: *read* (*tpsL2 t*) ! (*j3* + *2*) ≠ □ ⟷ *satisfies-clause ?a ?cl*
    using *tpsL2-def jk read-ncontents-eq-0*[*of tpsL2 t j3* + *2*] **by** *force*

  have *len*: *t* < *length* φ
    using *assms*(*2*) **by** (*simp add*: *formula-n-def*)
  have ∗∗: *sat-take* (*Suc t*) ⟷ *sat-take t* ∧ *satisfies-clause ?a ?cl*
    using *satisfies-take*[*OF len*] **by** *simp*

  show *tpsL3 t* = (*tpsL2 t*)[*j3* := (⌊*0*⌋$_N$, *1*)] **if** *read* (*tpsL2 t*) ! (*j3* + *2*) = □
  proof −
    have (*if sat-take* (*Suc t*) *then 1*::*nat else 0*) = *0*
      using *that* ∗ ∗∗ **by** *simp*
    then show *?thesis*
      unfolding *tpsL3-def tpsL2-def* **using** *that* **by** (*simp add*: *list-update-swap*)
  qed
  show *tpsL3 t* = (*tpsL2 t*) **if** *read* (*tpsL2 t*) ! (*j3* + *2*) ≠ □
  proof −
    have *sat-take t* = *sat-take* (*Suc t*)
      using ∗ ∗∗ *that* **by** *simp*
    then show *?thesis*
      unfolding *tpsL3-def tpsL2-def* **using** *that* **by** (*simp add*: *list-update-swap*)
  qed
qed

lemma *tmL3* [*transforms-intros*]:
  assumes *ttt* = *32* + *2* ∗ *nllength* (*formula-n* φ ! *t*) +
      79 ∗ (*nllength* (*formula-n* φ ! *t*))^2 +
      67 ∗ *nllength* (*formula-n* φ ! *t*) ∗ (*nllength vars*)^2
    and *t* < *length* (*formula-n* φ)
  shows *transforms tmL3* (*tpsL t*) *ttt* (*tpsL3 t*)
  unfolding *tmL3-def* **by** (*tform tps*: *assms*)

definition *tpsL4* :: *nat* ⇒ *tape list* **where**

tpsL4 t ≡ *tps0*
  [*j1* := *nlltape'* (*formula-n* φ) (*Suc t*),
   j3 := (⌊*sat-take* (*Suc t*)⌋$_B$, *1*),
   j3 + *1* := (⌊[]⌋$_{NL}$, *1*),
   j3 + *2* := (⌊*satisfies-clause* ($\lambda v.\ v \in$ *set vars*) (φ ! *t*)⌋$_B$, *1*)]

lemma *tmL4* [*transforms-intros*]:
  assumes *ttt* = *39* + *4* ∗ *nllength* (*formula-n* φ ! *t*) +
     79 ∗ (*nllength* (*formula-n* φ ! *t*))^2 +
     67 ∗ *nllength* (*formula-n* φ ! *t*) ∗ (*nllength vars*)^2
   and *t* < *length* (*formula-n* φ)
  shows *transforms tmL4* (*tpsL t*) *ttt* (*tpsL4 t*)
  unfolding *tmL4-def*
proof (*tform tps*: *assms*(*2*) *tps0 tpsL3-def tpsL4-def jk*)
  let *?zs* = *numlist* (*formula-n* φ ! *t*)
  have ∗: *tpsL3 t* ! (*j3* + *1*) = (⌊*formula-n* φ ! *t*⌋$_{NL}$, *1*)
   using *tpsL3-def jk* **by** *simp*
  then show *tpsL3 t* ::: (*j3* + *1*) = ⌊*?zs*⌋
   using *nlcontents-def* **by** *simp*
  show *proper-symbols ?zs*
   using *proper-symbols-numlist* **by** *simp*
  show *tpsL4 t* = (*tpsL3 t*)[*j3* + *1* := (⌊[]⌋, *1*)]
   unfolding *tpsL4-def tpsL3-def* **using** *nlcontents-Nil* **by** (*simp add*: *list-update-swap*)
  show *ttt* = *32* + *2* ∗ *nllength* (*formula-n* φ ! *t*) +
     79 ∗ (*nllength* (*formula-n* φ ! *t*))^2 +
     67 ∗ *nllength* (*formula-n* φ ! *t*) ∗ (*nllength vars*)^2 +
     (*tpsL3 t* :#: (*j3* + *1*) + *2* ∗ *length* (*numlist* (*formula-n* φ ! *t*)) + *6*)
   using ∗ *assms*(*1*) *nllength-def* **by** *simp*
qed

definition *tpsL5* :: *nat* ⇒ *tape list* **where**
  tpsL5 t ≡ *tps0*
  [*j1* := *nlltape'* (*formula-n* φ) (*Suc t*),
   j3 := (⌊*sat-take* (*Suc t*)⌋$_B$, *1*),
   j3 + *1* := (⌊[]⌋$_{NL}$, *1*),
   j3 + *2* := (⌊*0*⌋$_N$, *1*)]

lemma *tmL5*:
  assumes *ttt* = *49* + *4* ∗ *nllength* (*formula-n* φ ! *t*) +
     79 ∗ (*nllength* (*formula-n* φ ! *t*))^2 +
     67 ∗ *nllength* (*formula-n* φ ! *t*) ∗ (*nllength vars*)^2 +
     2 ∗ *nlength* (*if satisfies-clause* ($\lambda v.\ v \in$ *set vars*) (φ ! *t*) *then 1 else 0*)
   and *t* < *length* (*formula-n* φ)
  shows *transforms tmL5* (*tpsL t*) *ttt* (*tpsL5 t*)
  unfolding *tmL5-def* **by** (*tform tps*: *assms tps0 tpsL4-def tpsL5-def jk*)

definition *tpsL5'* :: *nat* ⇒ *tape list* **where**
  tpsL5' t ≡ *tps0*
  [*j1* := *nlltape'* (*formula-n* φ) (*Suc t*),
   j3 := (⌊*sat-take* (*Suc t*)⌋$_B$, *1*)]

lemma *tpsL5'*: *tpsL5'* = *tpsL5*
proof
  fix *t*
  have *5*: *j1* ≠ *j3* + *1*
   using *jk* **by** *simp*
  have *4*: *j3* ≠ *j3* + *1*
   by *simp*
  have *1*: *j3* ≠ *j3* + *2*
   by *simp*
  have *2*: *j3* + *1* ≠ *j3* + *2*
   by *simp*
  have *22*: *Suc j3* ≠ *Suc* (*Suc j3*)

by *simp*
have *3*: $j1 \neq j3 + 2$
using *jk* **by** *simp*
let *?tps1* = *tps0*
[$j1$:= *nlltape′* (*formula-n* φ) (*Suc t*)]
let *?tps2* = *tps0*
[$j1$:= *nlltape′* (*formula-n* φ) (*Suc t*),
$j3$:= ($\lfloor$*sat-take* (*Suc t*)$\rfloor_B$, *1*)]
have *tpsL5 t* = *tps0*
[$j1$:= *nlltape′* (*formula-n* φ) (*Suc t*),
$j3$:= ($\lfloor$*sat-take* (*Suc t*)$\rfloor_B$, *1*),
$j3 + 1$:= ($\lfloor[]\rfloor_{NL}$, *1*)]
unfolding *tpsL5-def*
using *tps0*(*5*)
list-update-swap[*OF 2*, *of ?tps2*]
list-update-swap[*OF 1*, *of ?tps1*]
list-update-swap[*OF 3*, *of tps0*]
list-update-id[*of tps0 j3 + 2*]
by (*simp only*:)
also have ... = *tps0*
[$j1$:= *nlltape′* (*formula-n* φ) (*Suc t*),
$j3$:= ($\lfloor$*sat-take* (*Suc t*)$\rfloor_B$, *1*)]
using *tps0*(*4*)
list-update-swap[*OF 4*, *of ?tps1*]
list-update-swap[*OF 5*, *of tps0*]
list-update-id[*of tps0 j3 + 1*]
by (*simp only*:)
finally show *tpsL5′ t* = *tpsL5 t*
using *tpsL5′-def* **by** *simp*
qed

lemma *tmL5′* [*transforms-intros*]:
assumes *ttt* = *51* + *83* ∗ (*nlllength* (*formula-n* φ))^2 +
67 ∗ *nlllength* (*formula-n* φ) ∗ (*nllength vars*)^2
and $t <$ *length* (*formula-n* φ)
shows *transforms tmL5* (*tpsL t*) *ttt* (*tpsL5′ t*)
proof −
let *?ttt* = *49* + *4* ∗ *nllength* (*formula-n* φ ! *t*) +
79 ∗ (*nllength* (*formula-n* φ ! *t*))^2 +
67 ∗ *nllength* (*formula-n* φ ! *t*) ∗ (*nllength vars*)^2 +
2 ∗ *nlength* (*if satisfies-clause* ($\lambda v.\ v \in$ *set vars*) (φ ! *t*) *then 1 else 0*)
have *?ttt* ≤ *49* + *4* ∗ *nllength* (*formula-n* φ ! *t*) +
79 ∗ (*nllength* (*formula-n* φ ! *t*))^2 +
67 ∗ *nllength* (*formula-n* φ ! *t*) ∗ (*nllength vars*)^2 +
2 ∗ *nlength 1*
by *simp*
also have ... = *51* + *4* ∗ *nllength* (*formula-n* φ ! *t*) +
79 ∗ (*nllength* (*formula-n* φ ! *t*))^2 +
67 ∗ *nllength* (*formula-n* φ ! *t*) ∗ (*nllength vars*)^2
using *nlength-1-simp* **by** *simp*
also have ... ≤ *51* + *4* ∗ *nlllength* (*formula-n* φ) +
79 ∗ (*nllength* (*formula-n* φ ! *t*))^2 +
67 ∗ *nllength* (*formula-n* φ ! *t*) ∗ (*nllength vars*)^2
using *member-le-nlllength-1*[*of formula-n* φ ! *t formula-n* φ] *assms*(*2*) **by** *simp*
also have ... ≤ *51* + *4* ∗ *nlllength* (*formula-n* φ) +
79 ∗ (*nlllength* (*formula-n* φ))^2 +
67 ∗ *nllength* (*formula-n* φ ! *t*) ∗ (*nllength vars*)^2
using *member-le-nlllength-1*[*of formula-n* φ ! *t formula-n* φ] *assms*(*2*) **by** *simp*
also have ... ≤ *51* + *4* ∗ *nlllength* (*formula-n* φ) +
79 ∗ (*nlllength* (*formula-n* φ))^2 +
67 ∗ *nlllength* (*formula-n* φ) ∗ (*nllength vars*)^2
using *member-le-nlllength-1*[*of formula-n* φ ! *t formula-n* φ] *assms*(*2*) **by** *auto*
also have ... ≤ *51* + *83* ∗ (*nlllength* (*formula-n* φ))^2 +

67 ∗ *nllength* (*formula-n* φ) ∗ (*nllength vars*)^2
    using *linear-le-pow* **by** *simp*
  finally have *?ttt* ≤ *ttt*
    using *assms*(*1*) **by** *simp*
  then show *?thesis*
    using *tpsL5′ transforms-monotone*[*OF tmL5*] *assms* **by** *simp*
qed

lemma *tmL* [*transforms-intros*]:
  assumes *ttt* = *length* (*formula-n* φ) ∗ (*53* + *83* ∗ (*nllength* (*formula-n* φ))^2 + *67* ∗ *nllength* (*formula-n* φ) ∗ (*nllength vars*)^2) + *1*
  shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsL* (*length* (*formula-n* φ)))
  unfolding *tmL-def*
proof (*tform*)
  let *?t* = *51* + *83* ∗ (*nllength* (*formula-n* φ))^2 + *67* ∗ *nllength* (*formula-n* φ) ∗ (*nllength vars*)^2
  have *tpsL5′ t* = *tpsL* (*Suc t*) **for** *t*
    using *tpsL5′-def tpsL-def* **by** *simp*
  then show ⋀*t. t* < *length* (*formula-n* φ) ⟹ *transforms tmL5* (*tpsL t*) *?t* (*tpsL* (*Suc t*))
    using *tmL5′* **by** *simp*

  let *?nss* = *formula-n* φ
  have ∗: *tpsL t* ! *j1* = *nlltape′ ?nss t* **for** *t*
    using *tpsL-def jk* **by** *simp*
  moreover have *read* (*tpsL t*) ! *j1* = *tpsL t* :.: *j1* **for** *t*
    using *tapes-at-read′*[*of j1 tpsL t*] *tpsL-def jk* **by** *simp*
  ultimately have *read* (*tpsL t*) ! *j1* = |.| (*nlltape′ ?nss t*) **for** *t*
    by *simp*
  then have *read* (*tpsL t*) ! *j1* = □ ⟷ (*t* ≥ *length ?nss*) **for** *t*
    using *nlltape′-tape-read* **by** *simp*
  then show
      ⋀*i. i* < *length ?nss* ⟹ *read* (*tpsL i*) ! *j1* ≠ □
      ¬ *read* (*tpsL* (*length ?nss*)) ! *j1* ≠ □
    using ∗ **by** *simp-all*
  show *length* (*formula-n* φ) ∗ (*?t* + *2*) + *1* ≤ *ttt*
    using *assms* **by** *simp*
qed

lemma *tm2*:
  assumes *ttt* = *length* (*formula-n* φ) ∗ (*53* + *83* ∗ (*nllength* (*formula-n* φ))^2 + *67* ∗ *nllength* (*formula-n* φ) ∗ (*nllength vars*)^2) + *13*
  shows *transforms tm2 tps0 ttt* (*tpsL* (*length* (*formula-n* φ)))
  unfolding *tm2-def*
proof (*tform tps*: *assms tps0 tpsL4-def tpsL5-def jk tpsL0*)
  show *transforms tm1 tps0 12* (*tpsL 0*)
    using *tm1 tpsL0* **by** *simp*
qed

definition *tps2* :: *tape list* **where**
  tps2 ≡ *tps0*
    [*j1* := *nlltape* (*formula-n* φ),
     j3 := (⌊(λ*v. v* ∈ *set vars*) ⊨ φ⌋$_B$, *1*)]

lemma *tps2*: *tps2* = *tpsL* (*length* (*formula-n* φ))
  using *formula-n-def tps2-def tpsL-def* **by** *simp*

lemma *tm2′*:
  assumes *ttt* = *length* (*formula-n* φ) ∗ (*53* + *83* ∗ (*nllength* (*formula-n* φ))^2 + *67* ∗ *nllength* (*formula-n* φ) ∗ (*nllength vars*)^2) + *13*
  shows *transforms tm2 tps0 ttt tps2*
  using *tm2 tps2 assms* **by** *simp*

end

end

lemma *transforms-tm-sat-formulaI* [*transforms-intros*]:
  fixes *j1 j2 j3* :: *tapeidx*
  fixes *tps tps'* :: *tape list* **and** *ttt k* :: *nat* **and** *vars* :: *nat list* **and** φ :: *formula*
  assumes $0 < j1$ $j1 \neq j2$ $j3 + 7 < k$ $j1 < j3$ $j2 < j3$ $0 < j2$ *length tps* $= k$
  assumes
    tps ! $j1$ = *nlltape'* (*formula-n* φ) 0
    tps ! $j2$ = *nltape'* *vars* 0
    tps ! $j3 = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j3 + 1) = (\lfloor [] \rfloor_{NL}, 1)$
    tps ! $(j3 + 2) = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j3 + 3) = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j3 + 4) = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j3 + 5) = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j3 + 6) = (\lfloor 0 \rfloor_N, 1)$
    tps ! $(j3 + 7) = (\lfloor 0 \rfloor_N, 1)$
  assumes *tps'* = *tps*
    [$j1$:= *nlltape* (*formula-n* φ),
     $j3 := (\lfloor (\lambda v.\ v \in \mathit{set\ vars}) \models \varphi \rfloor_B, 1)$]
  assumes *ttt* = *length* (*formula-n* φ) $*$ ($53 + 83 *$ (*nlllength* (*formula-n* φ))^2 + $67 *$ *nlllength* (*formula-n* φ) $*$ (*nllength vars*)^2) $+ 13$
  shows *transforms* (*tm-sat-formula j1 j2 j3*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-sat-formula j1 j2 j3* **.**
  show *?thesis*
    using *assms loc.tps2-def loc.tm2' loc.tm2-eq-tm-sat-formula* **by** *metis*
qed

4.2.3 A Turing machine for verifying SAT instances

The previous Turing machine, *tm-sat-formula*, expects a well-formed formula and a well-formed list representing an assignment on its tapes. The TM we ultimately need, however, is not guaranteed to be given anything well-formed as input and even the well-formed inputs require decoding from the binary alphabet to the quaternary alphabet used for lists of lists of numbers. The next TM takes care of all of that and, if everything was well-formed, runs *tm-sat-formula*. If the first element of the pair input is invalid, it outputs **1**, as required by the definition of SAT.

Thus, the next Turing machine implements the function *verify-sat* and therefore is a verifier for SAT.

definition *tm-verify-sat* :: *machine* **where**
  tm-verify-sat ≡
    tm-right-many $\{0..<22\}$;;
    tm-bindecode 0 2 ;;
    tm-unpair 2 3 4 ;;
    tm-even-length 3 5 ;;
    tm-proper-symbols-lt 3 6 4 ;;
    tm-and 6 5 ;;
    IF $\lambda rs.\ rs\ !\ 6 \neq \Box$ *THEN*
      tm-bindecode 3 7 ;;
      tm-numlistlist-wf 7 8 ;;
      IF $\lambda rs.\ rs\ !\ 8 \neq \Box$ *THEN*
        tm-proper-symbols-lt 4 10 4 ;;
        IF $\lambda rs.\ rs\ !\ 10 \neq \Box$ *THEN*
          tm-bindecode 4 11 ;;
          tm-rstrip ♯ *11* ;;
          tm-numlist-wf 11 12 ;;
          IF $\lambda rs.\ rs\ !\ 12 \neq \Box$ *THEN*
            tm-sat-formula 7 11 14 ;;
            tm-copyn 14 1
          ELSE
            []
          ENDIF
        ELSE

```
         []
       ENDIF
     ELSE
       tm-setn 1 1
     ENDIF
   ELSE
     tm-setn 1 1
   ENDIF
```

lemma *tm-verify-sat-tm*: *turing-machine 22 6 tm-verify-sat*
  unfolding *tm-verify-sat-def*
  using *tm-copyn-tm tm-setn-tm turing-machine-branch-turing-machine tm-sat-formula-tm tm-bindecode-tm tm-rstrip-tm tm-numlist-wf-tm tm-proper-symbols-lt-tm tm-numlistlist-wf-tm Nil-tm tm-right-many-tm tm-unpair-tm tm-even-length-tm tm-and-tm*
  by *simp*

locale *turing-machine-verify-sat*
begin

definition *tm1* ≡ *tm-right-many* {*0..<22*}
definition *tm2* ≡ *tm1* ;; *tm-bindecode 0 2*
definition *tm3* ≡ *tm2* ;; *tm-unpair 2 3 4*
definition *tm4* ≡ *tm3* ;; *tm-even-length 3 5*
definition *tm5* ≡ *tm4* ;; *tm-proper-symbols-lt 3 6 4*
definition *tm6* ≡ *tm5* ;; *tm-and 6 5*

definition *tmTTT1* ≡ *tm-bindecode 4 11*
definition *tmTTT2* ≡ *tmTTT1* ;; *tm-rstrip* ♯ *11*
definition *tmTTT3* ≡ *tmTTT2* ;; *tm-numlist-wf 11 12*

definition *tmTTTT1* ≡ *tm-sat-formula 7 11 14*
definition *tmTTTT2* ≡ *tmTTTT1* ;; *tm-copyn 14 1*
definition *tmTTTI* ≡ *IF* λ*rs. rs* ! *12* ≠ □ *THEN tmTTTT2 ELSE* [] *ENDIF*

definition *tmTTT* ≡ *tmTTT3* ;; *tmTTTI*
definition *tmTTI* ≡ *IF* λ*rs. rs* ! *10* ≠ □ *THEN tmTTT ELSE* [] *ENDIF*

definition *tmTT1* ≡ *tm-proper-symbols-lt 4 10 4*
definition *tmTT* ≡ *tmTT1* ;; *tmTTI*
definition *tmTI* ≡ *IF* λ*rs. rs* ! *8* ≠ □ *THEN tmTT ELSE tm-setn 1 1 ENDIF*

definition *tmT1* ≡ *tm-bindecode 3 7*
definition *tmT2* ≡ *tmT1* ;; *tm-numlistlist-wf 7 8*
definition *tmT* ≡ *tmT2* ;; *tmTI*

definition *tmI* ≡ *IF* λ*rs. rs* ! *6* ≠ □ *THEN tmT ELSE tm-setn 1 1 ENDIF*
definition *tm7* ≡ *tm6* ;; *tmI*

lemma *tm7-eq-tm-verify-sat*: *tm7* = *tm-verify-sat*
  unfolding *tm-verify-sat-def tm7-def tmI-def tmT-def tmT2-def tmTI-def tmT1-def tmTT-def tmTT1-def tmTTI-def tmTTT-def tmTTT3-def tmTTTT1-def tmTTTI-def tmTTTT2-def tmTTT3-def tmTTT2-def tmTTT1-def tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def*
  by *simp*

context
  fixes *tps0* :: *tape list* **and** *zs* :: *symbol list*
  assumes *zs*: *bit-symbols zs*
  assumes *tps0*: *tps0* = *snd* (*start-config 22 zs*)
begin

definition *tps1* ≡ *map* (λ*tp. tp* |#=| *1*) *tps0*

lemma *map-upt-length*: *map f xs* = *map* (λi. *f* (*xs* ! *i*)) [*0*..<*length xs*]
  by (*smt* (*verit*, *ccfv-SIG*) *in-set-conv-nth length-map map-eq-conv map-nth nth-map*)

lemma *tps1*:
  tps1 ! *0* = ($\lfloor$*zs*$\rfloor$, *1*)
  0 < *j* $\Longrightarrow$ *j* < *22* $\Longrightarrow$ *tps1* ! *j* = ($\lfloor$[]$\rfloor$, *1*)
  length tps1 = *22*
  using *tps0 start-config-def tps1-def* **by** *auto*

lemma *tm1* [*transforms-intros*]: *transforms tm1 tps0 1 tps1*
  unfolding *tm1-def*
proof (*tform tps*: *tps0 tps1-def*)
  have *length tps0* = *22*
    using *tps0 start-config-def* **by** *simp*
  then have *map* (λj. *if j* $\in$ {*0*..<*22*} *then tps0* ! *j* |+| *1 else tps0* ! *j*) [*0*..<*length tps0*] =
      map (λj. *tps0* ! *j* |+| *1*) [*0*..<*length tps0*]
    by *simp*
  also have ... = *map* (λj. *tps0* ! *j* |#=| *1*) [*0*..<*length tps0*]
    using *tps0* ‹*length tps0* = *22*› *start-config-pos* **by** *simp*
  also have ... = *map* (λtp. *tp* |#=| *1*) *tps0*
    using *map-upt-length*[*of* λtp. *tp* |#=| *1 tps0*] **by** *simp*
  also have ... = *tps1*
    using *tps1-def* **by** *simp*
  finally show *tps1* = *map* (λj. *if j* $\in$ {*0*..<*22*} *then tps0* ! *j* |+| *1 else tps0* ! *j*) [*0*..<*length tps0*]
    by *simp*
qed

definition *tps2* $\equiv$ *tps1*
  [*2* := ($\lfloor$*bindecode zs*$\rfloor$, *1*)]

lemma *tm2* [*transforms-intros*]:
  assumes *ttt* = *8* + *3* $*$ *length zs*
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def* **by** (*tform tps*: *assms zs tps1 tps2-def*)

definition *tps3* $\equiv$ *tps1*
  [*2* := ($\lfloor$*bindecode zs*$\rfloor$, *1*),
   3 := ($\lfloor$*first* (*bindecode zs*)$\rfloor$, *1*),
   4 := ($\lfloor$*second* (*bindecode zs*)$\rfloor$, *1*)]

lemma *tm3* [*transforms-intros*]:
  assumes *ttt* = *21* + *3* $*$ *length zs* + *6* $*$ *length* (*bindecode zs*)
  shows *transforms tm3 tps0 ttt tps3*
  unfolding *tm3-def*
proof (*tform tps*: *assms zs tps2-def tps1 tps3-def*)
  show *proper-symbols* (*bindecode zs*)
    using *zs proper-bindecode* **by** *simp*
  show *ttt* = *8* + *3* $*$ *length zs* + (*6* $*$ *length* (*bindecode zs*) + *13*)
    using *assms* **by** *simp*
qed

definition *tps4* $\equiv$ *tps1*
  [*2* := ($\lfloor$*bindecode zs*$\rfloor$, *1*),
   3 := ($\lfloor$*first* (*bindecode zs*)$\rfloor$, *1*),
   4 := ($\lfloor$*second* (*bindecode zs*)$\rfloor$, *1*),
   5 := ($\lfloor$*even* (*length* (*first* (*bindecode zs*)))$\rfloor_B$, *1*)]

lemma *tm4* [*transforms-intros*]:
  assumes *ttt* = *28* + *3* $*$ *length zs* + *6* $*$ *length* (*bindecode zs*) + *7* $*$ *length* (*first* (*bindecode zs*))
  shows *transforms tm4 tps0 ttt tps4*
  unfolding *tm4-def*
proof (*tform tps*: *assms zs tps1 tps3-def tps4-def*)
  show *proper-symbols* (*first* (*bindecode zs*))

using *zs proper-bindecode first-def* **by** *simp*
show *tps3* ! *5* = ($\lfloor 0 \rfloor_N$, *1*)
using *tps3-def canrepr-0 tps1* **by** *simp*
qed

definition *tps5* ≡ *tps1*
[*2* := (⌊*bindecode zs*⌋, *1*),
3 := (⌊*first* (*bindecode zs*)⌋, *1*),
4 := (⌊*second* (*bindecode zs*)⌋, *1*),
5 := ($\lfloor$*even* (*length* (*first* (*bindecode zs*)))$\rfloor_B$, *1*),
6 := ($\lfloor$*proper-symbols-lt 4* (*first* (*bindecode zs*))$\rfloor_B$, *1*)]

lemma *tm5* [*transforms-intros*]:
assumes *ttt* = *33* + *3* ∗ *length zs* + *6* ∗ *length* (*bindecode zs*) + *14* ∗ *length* (*first* (*bindecode zs*))
shows *transforms tm5 tps0 ttt tps5*
unfolding *tm5-def*
proof (*tform tps*: *assms zs tps1 tps4-def tps5-def*)
show *proper-symbols* (*first* (*bindecode zs*))
using *zs proper-bindecode first-def* **by** *simp*
qed

abbreviation *ys* ≡ *bindecode zs*
abbreviation *xs* ≡ *bindecode* (*first ys*)
abbreviation *vs* ≡ *rstrip 5* (*bindecode* (*second ys*))

definition *tps6* ≡ *tps1*
[*2* := (⌊*ys*⌋, *1*),
3 := (⌊*first ys*⌋, *1*),
4 := (⌊*second ys*⌋, *1*),
5 := ($\lfloor$*even* (*length* (*first ys*))$\rfloor_B$, *1*),
6 := ($\lfloor$*proper-symbols-lt 4* (*first ys*) ∧ *even* (*length* (*first ys*))$\rfloor_B$, *1*)]

lemma *tm6* [*transforms-intros*]:
assumes *ttt* = *36* + *3* ∗ *length zs* + *6* ∗ *length* (*bindecode zs*) + *14* ∗ *length* (*first* (*bindecode zs*))
shows *transforms tm6 tps0 ttt tps6*
unfolding *tm6-def* **by** (*tform tps*: *assms zs tps1 tps5-def tps6-def*)

context
assumes *bs-even*: *proper-symbols-lt 4* (*first ys*) ∧ *even* (*length* (*first ys*))
begin

lemma *bs*: *bit-symbols* (*first ys*)
using *bs-even* **by** *fastforce*

definition *tpsT1* ≡ *tps1*
[*2* := (⌊*ys*⌋, *1*),
3 := (⌊*first ys*⌋, *1*),
4 := (⌊*second ys*⌋, *1*),
5 := ($\lfloor$*even* (*length* (*first ys*))$\rfloor_B$, *1*),
6 := ($\lfloor$*proper-symbols-lt 4* (*first ys*) ∧ *even* (*length* (*first ys*))$\rfloor_B$, *1*),
7 := (⌊*bindecode* (*first ys*)⌋, *1*)]

lemma *tmT1* [*transforms-intros*]:
assumes *ttt* = *7* + *3* ∗ *length* (*first ys*)
shows *transforms tmT1 tps6 ttt tpsT1*
unfolding *tmT1-def* **by** (*tform tps*: *assms bs tps1 tps6-def tpsT1-def*)

definition *tpsT2* ≡ *tps1*
[*2* := (⌊*ys*⌋, *1*),
3 := (⌊*first ys*⌋, *1*),
4 := (⌊*second ys*⌋, *1*),
5 := ($\lfloor$*even* (*length* (*first ys*))$\rfloor_B$, *1*),
6 := ($\lfloor$*proper-symbols-lt 4* (*first ys*) ∧ *even* (*length* (*first ys*))$\rfloor_B$, *1*),

7 := (⌊*bindecode* (*first ys*)⌋, *1*),
8 := (⌊*numlistlist-wf* (*bindecode* (*first ys*))⌋$_B$, *1*)]

lemma *tmT2* [*transforms-intros*]:
assumes *ttt* = *213* + *3* ∗ *length* (*first ys*) + *39* ∗ *length* (*bindecode* (*first ys*))
shows *transforms tmT2 tps6 ttt tpsT2*
unfolding *tmT2-def*
proof (*tform tps*: *assms tps1 tpsT1-def tpsT2-def*)
show *proper-symbols* (*bindecode* (*first ys*))
using *proper-bindecode* **by** *simp*
show *ttt* = *7* + *3* ∗ *length* (*first ys*) + (*206* + *39* ∗ *length* (*bindecode* (*first ys*)))
using *assms* **by** *simp*
qed

context
assumes *first-wf*: *numlistlist-wf* (*bindecode* (*first ys*))
begin

definition *tpsTT1* ≡ *tps1*
[*2* := (⌊*ys*⌋, *1*),
3 := (⌊*first ys*⌋, *1*),
4 := (⌊*second ys*⌋, *1*),
5 := (⌊*even* (*length* (*first ys*))⌋$_B$, *1*),
6 := (⌊*proper-symbols-lt 4* (*first ys*) ∧ *even* (*length* (*first ys*))⌋$_B$, *1*),
7 := (⌊*bindecode* (*first ys*)⌋, *1*),
8 := (⌊*numlistlist-wf* (*bindecode* (*first ys*))⌋$_B$, *1*),
10 := (⌊*proper-symbols-lt 4* (*second ys*)⌋$_B$, *1*)]

lemma *tmTT1* [*transforms-intros*]:
assumes *ttt* = *5* + *7* ∗ *length* (*second ys*)
shows *transforms tmTT1 tpsT2 ttt tpsTT1*
unfolding *tmTT1-def*
proof (*tform tps*: *tps1 tpsT2-def tpsTT1-def assms*)
show *proper-symbols* (*second ys*)
using *proper-bindecode second-def zs* **by** *simp*
qed

context
assumes *proper-second*: *proper-symbols-lt 4* (*second ys*)
begin

definition *tpsTTT1* ≡ *tps1*
[*2* := (⌊*ys*⌋, *1*),
3 := (⌊*first ys*⌋, *1*),
4 := (⌊*second ys*⌋, *1*),
5 := (⌊*even* (*length* (*first ys*))⌋$_B$, *1*),
6 := (⌊*proper-symbols-lt 4* (*first ys*) ∧ *even* (*length* (*first ys*))⌋$_B$, *1*),
7 := (⌊*xs*⌋, *1*),
8 := (⌊*numlistlist-wf xs*⌋$_B$, *1*),
10 := (⌊*proper-symbols-lt 4* (*second ys*)⌋$_B$, *1*),
11 := (⌊*bindecode* (*second ys*)⌋, *1*)]

lemma *tmTTT1* [*transforms-intros*]:
assumes *ttt* = *7* + *3* ∗ *length* (*second ys*)
shows *transforms tmTTT1 tpsTT1 ttt tpsTTT1*
unfolding *tmTTT1-def*
proof (*tform tps*: *assms tps1 tpsT2-def tpsTT1-def tpsTTT1-def*)
show *bit-symbols* (*second ys*)
using *proper-second* **by** *fastforce*
qed

definition *tpsTTT2* ≡ *tps1*
[*2* := (⌊*ys*⌋, *1*),

$3 := (\lfloor \mathit{first\ ys} \rfloor, 1)$,
$4 := (\lfloor \mathit{second\ ys} \rfloor, 1)$,
$5 := (\lfloor \mathit{even}\ (\mathit{length}\ (\mathit{first\ ys})) \rfloor_B, 1)$,
$6 := (\lfloor \mathit{proper\text{-}symbols\text{-}lt}\ 4\ (\mathit{first\ ys}) \land \mathit{even}\ (\mathit{length}\ (\mathit{first\ ys})) \rfloor_B, 1)$,
$7 := (\lfloor \mathit{xs} \rfloor, 1)$,
$8 := (\lfloor \mathit{numlistlist\text{-}wf\ xs} \rfloor_B, 1)$,
$10 := (\lfloor \mathit{proper\text{-}symbols\text{-}lt}\ 4\ (\mathit{second\ ys}) \rfloor_B, 1)$,
$11 := (\lfloor \mathit{vs} \rfloor, 1)]$

lemma *tmTTT2* [*transforms-intros*]:
assumes $\mathit{ttt} = 12 + 3 * \mathit{length}\ (\mathit{second\ ys}) + 3 * \mathit{length}\ (\mathit{bindecode}\ (\mathit{second\ ys}))$
shows *transforms tmTTT2 tpsTT1 ttt tpsTTT2*
unfolding *tmTTT2-def*
proof (*tform tps*: *assms tps1 tpsTTT1-def tpsTTT2-def*)
show *proper-symbols* (*bindecode* (*second ys*))
using *proper-bindecode* **by** *simp*
show $\mathit{ttt} = 7 + 3 * \mathit{length}\ (\mathit{second\ ys}) + (3 * \mathit{length}\ (\mathit{bindecode}\ (\mathit{second\ ys})) + 5)$
using *assms* **by** *simp*
qed

definition $\mathit{tpsTTT3} \equiv \mathit{tps1}$
$[2 := (\lfloor \mathit{ys} \rfloor, 1)$,
$3 := (\lfloor \mathit{first\ ys} \rfloor, 1)$,
$4 := (\lfloor \mathit{second\ ys} \rfloor, 1)$,
$5 := (\lfloor \mathit{even}\ (\mathit{length}\ (\mathit{first\ ys})) \rfloor_B, 1)$,
$6 := (\lfloor \mathit{proper\text{-}symbols\text{-}lt}\ 4\ (\mathit{first\ ys}) \land \mathit{even}\ (\mathit{length}\ (\mathit{first\ ys})) \rfloor_B, 1)$,
$7 := (\lfloor \mathit{xs} \rfloor, 1)$,
$8 := (\lfloor \mathit{numlistlist\text{-}wf\ xs} \rfloor_B, 1)$,
$10 := (\lfloor \mathit{proper\text{-}symbols\text{-}lt}\ 4\ (\mathit{second\ ys}) \rfloor_B, 1)$,
$11 := (\lfloor \mathit{vs} \rfloor, 1)$,
$12 := (\lfloor \mathit{numlist\text{-}wf\ vs} \rfloor_B, 1)]$

lemma *tmTTT3* [*transforms-intros*]:
assumes $\mathit{ttt} = 106 + 3 * \mathit{length}\ (\mathit{second\ ys}) + 3 * \mathit{length}\ (\mathit{bindecode}\ (\mathit{second\ ys})) + 19 * \mathit{length\ vs}$
shows *transforms tmTTT3 tpsTT1 ttt tpsTTT3*
unfolding *tmTTT3-def*
proof (*tform tps*: *assms tps1 tpsTTT2-def tpsTTT3-def*)
show *proper-symbols vs*
using *proper-bindecode rstrip-def* **by** *simp*
qed

context
assumes *second-wf*: *numlist-wf vs*
begin

definition $\mathit{tpsTTTT1} \equiv \mathit{tps1}$
$[2 := (\lfloor \mathit{ys} \rfloor, 1)$,
$3 := (\lfloor \mathit{first\ ys} \rfloor, 1)$,
$4 := (\lfloor \mathit{second\ ys} \rfloor, 1)$,
$5 := (\lfloor \mathit{even}\ (\mathit{length}\ (\mathit{first\ ys})) \rfloor_B, 1)$,
$6 := (\lfloor \mathit{proper\text{-}symbols\text{-}lt}\ 4\ (\mathit{first\ ys}) \land \mathit{even}\ (\mathit{length}\ (\mathit{first\ ys})) \rfloor_B, 1)$,
$7 := (\lfloor \mathit{xs} \rfloor, 1)$,
$8 := (\lfloor \mathit{numlistlist\text{-}wf\ xs} \rfloor_B, 1)$,
$10 := (\lfloor \mathit{proper\text{-}symbols\text{-}lt}\ 4\ (\mathit{second\ ys}) \rfloor_B, 1)$,
$11 := (\lfloor \mathit{vs} \rfloor, 1)$,
$12 := (\lfloor \mathit{numlist\text{-}wf\ vs} \rfloor_B, 1)$,
$7 := \mathit{nlltape}\ (\mathit{formula\text{-}n}\ (\mathit{zs\text{-}formula\ xs}))$,
$14 := (\lfloor (\lambda v.\ v \in \mathit{set}\ (\mathit{zs\text{-}numlist\ vs})) \models \mathit{zs\text{-}formula\ xs} \rfloor_B, 1)]$

lemma *tmTTTT1* [*transforms-intros*]:
assumes $\mathit{ttt} = \mathit{length}\ (\mathit{formula\text{-}n}\ (\mathit{zs\text{-}formula\ xs})) *$
$(53 + 83 * (\mathit{nlllength}\ (\mathit{formula\text{-}n}\ (\mathit{zs\text{-}formula\ xs})))^2 + 67 * \mathit{nlllength}\ (\mathit{formula\text{-}n}\ (\mathit{zs\text{-}formula\ xs})) * (\mathit{nllength}$
$(\mathit{zs\text{-}numlist\ vs}))^2) +$

13
shows *transforms tmTTTT1 tpsTTT3 ttt tpsTTTT1*
unfolding *tmTTTT1-def*
proof (*tform time*: *assms*)
show
$tpsTTT3 ! 14 = (\lfloor 0 \rfloor_N, 1)$
$tpsTTT3 ! (14 + 2) = (\lfloor 0 \rfloor_N, 1)$
$tpsTTT3 ! (14 + 3) = (\lfloor 0 \rfloor_N, 1)$
$tpsTTT3 ! (14 + 4) = (\lfloor 0 \rfloor_N, 1)$
$tpsTTT3 ! (14 + 5) = (\lfloor 0 \rfloor_N, 1)$
$tpsTTT3 ! (14 + 6) = (\lfloor 0 \rfloor_N, 1)$
$tpsTTT3 ! (14 + 7) = (\lfloor 0 \rfloor_N, 1)$
unfolding *tpsTTT3-def* **using** *tps1 canrepr-0* **by** *auto*
show $tpsTTT3 ! (14 + 1) = (\lfloor [] \rfloor_{NL}, 1)$
unfolding *tpsTTT3-def* **using** *tps1 nlcontents-Nil* **by** *simp*
show $14 + 7 < length\ tpsTTT3$
unfolding *tpsTTT3-def* **using** *tps1* **by** *simp*
let *?phi* = *zs-formula xs*
have *numlistlist* (*formula-n ?phi*) = *xs*
using *formula-zs-def formula-zs-formula first-wf* **by** *simp*
then have $nlltape'$ (*formula-n ?phi*) $0 = (\lfloor xs \rfloor, 1)$
by (*simp add*: *nllcontents-def*)
then show $tpsTTT3 ! 7 = nlltape'$ (*formula-n ?phi*) *0*
unfolding *tpsTTT3-def* **using** *tps1* **by** *simp*
let *?vars* = *zs-numlist vs*
have *numlist ?vars* = *vs*
using *numlist-zs-numlist second-wf* **by** *simp*
then have $nltape'\ ?vars\ 0 = (\lfloor vs \rfloor, 1)$
by (*simp add*: *nlcontents-def*)
then show $tpsTTT3 ! 11 = nltape'\ ?vars\ 0$
unfolding *tpsTTT3-def* **using** *tps1* **by** *simp*
show *tpsTTTT1* = *tpsTTT3*
[*7* := *nlltape* (*formula-n* (*zs-formula xs*)),
14 := $(\lfloor (\lambda v.\ v \in set\ (zs\text{-}numlist\ vs)) \models zs\text{-}formula\ xs \rfloor_B, 1)$]
unfolding *tpsTTTT1-def tpsTTT3-def* **by** *fast*
qed

definition *tpsTTTT2* $\equiv$ *tps1*
[$1 := (\lfloor (\lambda v.\ v \in set\ (zs\text{-}numlist\ vs)) \models zs\text{-}formula\ xs \rfloor_B, 1)$,
$2 := (\lfloor ys \rfloor, 1)$,
$3 := (\lfloor first\ ys \rfloor, 1)$,
$4 := (\lfloor second\ ys \rfloor, 1)$,
$5 := (\lfloor even\ (length\ (first\ ys)) \rfloor_B, 1)$,
$6 := (\lfloor proper\text{-}symbols\text{-}lt\ 4\ (first\ ys) \wedge even\ (length\ (first\ ys)) \rfloor_B, 1)$,
$7 := (\lfloor xs \rfloor, 1)$,
$8 := (\lfloor numlistlist\text{-}wf\ xs \rfloor_B, 1)$,
$10 := (\lfloor proper\text{-}symbols\text{-}lt\ 4\ (second\ ys) \rfloor_B, 1)$,
$11 := (\lfloor vs \rfloor, 1)$,
$12 := (\lfloor numlist\text{-}wf\ vs \rfloor_B, 1)$,
7 := *nlltape* (*formula-n* (*zs-formula xs*)),
$14 := (\lfloor (\lambda v.\ v \in set\ (zs\text{-}numlist\ vs)) \models zs\text{-}formula\ xs \rfloor_B, 1)$]

lemma *tmTTTT2*:
assumes $ttt = length\ (formula\text{-}n\ (zs\text{-}formula\ xs)) *$
$(53 + 83 * (nlllength\ (formula\text{-}n\ (zs\text{-}formula\ xs)))^2 + 67 * nlllength\ (formula\text{-}n\ (zs\text{-}formula\ xs)) * (nllength\ (zs\text{-}numlist\ vs))^2) +$
$27 + 3 * (nlength\ (if\ (\lambda v.\ v \in set\ (zs\text{-}numlist\ vs)) \models zs\text{-}formula\ xs\ then\ 1\ else\ 0))$
shows *transforms tmTTTT2 tpsTTT3 ttt tpsTTTT2*
unfolding *tmTTTT2-def*
proof (*tform*)
show $14 < length\ tpsTTTT1\ 1 < length\ tpsTTTT1$
unfolding *tpsTTTT1-def* **using** *tps1* **by** *simp-all*
show $tpsTTTT1 ! 1 = (\lfloor 0 \rfloor_N, 1)$

unfolding *tpsTTTT1-def* **using** *tps1 canrepr-0* **by** *auto*
let *?b* = *if* ($\lambda v.\ v \in$ *set* (*zs-numlist vs*)) $\models$ *zs-formula xs then 1 else 0* :: *nat*
show *tpsTTTT1* ! *14* = ($\lfloor ?b \rfloor_N$, *1*)
unfolding *tpsTTTT1-def* **using** *tps1* **by** *simp*
show *ttt* = *length* (*formula-n* (*zs-formula xs*)) $*$
($53 + 83 *$ (*nlllength* (*formula-n* (*zs-formula xs*)))^2 +
$67 *$ *nlllength* (*formula-n* (*zs-formula xs*)) $*$ (*nllength* (*zs-numlist vs*))^2) +
$13 + (14 + 3 *$
(*nlength* (*if* ($\lambda v.\ v \in$ *set* (*zs-numlist vs*)) $\models$ *zs-formula xs then 1 else 0*) + *nlength 0*))
using *assms* **by** *simp*
show *tpsTTTT2* = *tpsTTTT1*
[*1* := ($\lfloor(\lambda v.\ v \in$ *set* (*zs-numlist vs*)) $\models$ *zs-formula xs*$\rfloor_B$, *1*)]
unfolding *tpsTTTT2-def tpsTTTT1-def* **by** (*simp add*: *list-update-swap*)
qed

lemma *tmTTTT2'* [*transforms-intros*]:
assumes *ttt* = $203 *$ *length zs* ^ *4* + *30*
shows *transforms tmTTTT2 tpsTTT3 ttt tpsTTTT2*
proof −
let *?phi* = *zs-formula xs*
let *?ttt* = *length* (*formula-n ?phi*) $*$
($53 + 83 *$ (*nlllength* (*formula-n ?phi*))^2 + $67 *$ *nlllength* (*formula-n ?phi*) $*$ (*nllength* (*zs-numlist vs*))^2) +
$27 + 3 *$ (*nlength* (*if* ($\lambda v.\ v \in$ *set* (*zs-numlist vs*)) $\models$ *?phi then 1 else 0*))
have *nlllength* (*formula-n ?phi*) $\leq$ *length xs*
using *formula-zs-def formula-zs-formula first-wf nlllength-def* **by** *simp*
then have *1*: *nlllength* (*formula-n ?phi*) $\leq$ *length zs*
by (*metis div-le-dividend le-trans length-bindecode length-first*)
moreover have *length* (*formula-n ?phi*) $\leq$ *nlllength* (*formula-n ?phi*)
by (*simp add*: *length-le-nlllength*)
ultimately have *2*: *length* (*formula-n ?phi*) $\leq$ *length zs*
by *simp*
have *nllength* (*zs-numlist vs*) $\leq$ *length vs*
using *second-wf numlist-zs-numlist nllength-def* **by** *simp*
moreover have *length vs* $\leq$ *length zs*
using *second-def length-bindecode length-rstrip-le* **by** (*metis div-le-dividend dual-order.trans length-second*)
ultimately have *3*: *nllength* (*zs-numlist vs*) $\leq$ *length zs*
by *simp*
have *4*: *nlength* (*if* ($\lambda v.\ v \in$ *set* (*zs-numlist vs*)) $\models$ *?phi then 1 else 0*) $\leq$ *1*
using *nlength-1-simp* **by** *simp*

have *?ttt* $\leq$ *length* (*formula-n ?phi*) $*$
($53 + 83 *$ (*nlllength* (*formula-n ?phi*))^2 + $67 *$ *nlllength* (*formula-n ?phi*) $*$ (*nllength* (*zs-numlist vs*))^2)
+ *30*
using *4* **by** *simp*
also have ... $\leq$ *length zs* $*$
($53 + 83 *$ (*nlllength* (*formula-n ?phi*))^2 + $67 *$ *nlllength* (*formula-n ?phi*) $*$ (*nllength* (*zs-numlist vs*))^2)
+ *30*
using *2* **by** *simp*
also have ... $\leq$ *length zs* $*$ ($53 + 83 *$ (*length zs*)^2 + $67 *$ *length zs* $*$ (*nllength* (*zs-numlist vs*))^2) + *30*
using *1* **by** (*simp add*: *add-mono*)
also have ... $\leq$ *length zs* $*$ ($53 + 83 *$ (*length zs*)^2 + $67 *$ *length zs* $*$ (*length zs*)^2) + *30*
using *3* **by** *simp*
also have ... = $53 *$ *length zs* + $83 *$ *length zs* ^ *3* + $67 *$ *length zs* ^ *4* + *30*
by *algebra*
also have ... $\leq$ $53 *$ *length zs* + $83 *$ *length zs* ^ *4* + $67 *$ *length zs* ^ *4* + *30*
using *pow-mono'* **by** *simp*
also have ... $\leq$ $53 *$ *length zs* ^ *4* + $83 *$ *length zs* ^ *4* + $67 *$ *length zs* ^ *4* + *30*
using *linear-le-pow* **by** *simp*
also have ... = $203 *$ *length zs* ^ *4* + *30*
by *simp*
finally have *?ttt* $\leq$ $203 *$ *length zs* ^ *4* + *30* **.**
then show *?thesis*
using *assms tmTTTT2 transforms-monotone* **by** *simp*

qed

end

definition *tpsTTT* $\equiv$ (*if numlist-wf vs then tpsTTTT2 else tpsTTT3*)

lemma *length-tpsTTT*: *length tpsTTT* = *22*
  using *tpsTTT-def tpsTTTT2-def tpsTTT3-def tps1* **by** (*metis* (*no-types*, *lifting*) *length-list-update*)

lemma *tpsTTT*: *tpsTTT* ! *1* =
  ($\lfloor$*if numlist-wf vs then* (*if* (λ*v*. *v* $\in$ *set* (*zs-numlist vs*)) $\models$ *zs-formula xs then 1 else 0*) *else 0*$\rfloor_N$, *1*)
proof (*cases numlist-wf vs*)
  case *True*
  then have *tpsTTT* ! *1* = *tpsTTTT2* ! *1*
    using *tpsTTT-def* **by** *simp*
  also have ... = ($\lfloor$(λ*v*. *v* $\in$ *set* (*zs-numlist vs*)) $\models$ *zs-formula xs*$\rfloor_B$, *1*)
    unfolding *tpsTTTT2-def*[*OF True*] **using** *tps1* **by** *simp*
  finally show *?thesis*
    using *True* **by** *simp*
next
  case *False*
  then have *tpsTTT* ! *1* = *tpsTTT3* ! *1*
    using *tpsTTT-def* **by** *simp*
  also have ... = ($\lfloor$*0*$\rfloor_N$, *1*)
    unfolding *tpsTTT3-def* **using** *tps1 canrepr-0* **by** *simp*
  finally show *?thesis*
    using *False* **by** *simp*
qed

lemma *tmTTTI* [*transforms-intros*]:
  assumes *ttt* = *203* $*$ *length zs* ^ *4* + *32*
  shows *transforms tmTTTI tpsTTT3 ttt tpsTTT*
  unfolding *tmTTTI-def*
proof (*tform time*: *assms*)
  have $*$: *read tpsTTT3* ! *12* $\neq$ $\Box$ $\longleftrightarrow$ *numlist-wf vs*
    using *tpsTTT3-def tps1 read-ncontents-eq-0* **by** *simp*
  show *read tpsTTT3* ! *12* $\neq$ $\Box$ $\Longrightarrow$ *numlist-wf vs*
    using $*$ **by** *simp*
  show *read tpsTTT3* ! *12* $\neq$ $\Box$ $\Longrightarrow$ *tpsTTT* = *tpsTTTT2*
    using $*$ *tpsTTT-def* **by** *simp*
  show $\neg$ *read tpsTTT3* ! *12* $\neq$ $\Box$ $\Longrightarrow$ *tpsTTT* = *tpsTTT3*
    using $*$ *tpsTTT-def* **by** *simp*
qed

lemma *tmTTT*:
  assumes *ttt* = *138* + *3* $*$ *length* (*second ys*) + *3* $*$ *length* (*bindecode* (*second ys*)) +
      19 $*$ *length vs* + *203* $*$ *length zs* ^ *4*
  shows *transforms tmTTT tpsTT1 ttt tpsTTT*
  unfolding *tmTTT-def* **by** (*tform tps*: *assms*)

lemma *tmTTT'* [*transforms-intros*]:
  assumes *ttt* = *138* + *228* $*$ *length zs* ^ *4*
  shows *transforms tmTTT tpsTT1 ttt tpsTTT*
proof −
  let *?ttt* = *138* + *3* $*$ *length* (*second ys*) + *3* $*$ *length* (*bindecode* (*second ys*)) +
     19 $*$ *length vs* + *203* $*$ *length zs* ^ *4*
  have *length ys* $\leq$ *length zs*
    by *simp*
  then have *1*: *length* (*second ys*) $\leq$ *length zs*
    using *length-second dual-order.trans* **by** *blast*
  then have *2*: *length* (*bindecode* (*second ys*)) $\leq$ *length zs*
    by *simp*
  then have *3*: *length vs* $\leq$ *length zs*

by (*meson dual-order.trans length-rstrip-le*)

have *?ttt* ≤ *138* + *3* ∗ *length zs* + *3* ∗ *length zs* + *19* ∗ *length zs* + *203* ∗ *length zs* ^ *4*
using *1 2 3* **by** *simp*
also have ... = *138* + *25* ∗ *length zs* + *203* ∗ *length zs* ^ *4*
by *simp*
also have ... ≤ *138* + *25* ∗ *length zs* ^ *4* + *203* ∗ *length zs* ^ *4*
using *linear-le-pow* **by** *simp*
also have ... = *138* + *228* ∗ *length zs* ^ *4*
by *simp*
finally have *?ttt* ≤ *138* + *228* ∗ *length zs* ^ *4* **.**
then show *?thesis*
using *assms tmTTT transforms-monotone* **by** *blast*
qed

end

definition *tpsTT* ≡ (*if proper-symbols-lt 4* (*second ys*) *then tpsTTT else tpsTT1*)

lemma *length-tpsTT*: *length tpsTT* = *22*
using *tpsTT-def length-tpsTTT tpsTT1-def tps1* **by** *simp*

lemma *tpsTT*: *tpsTT* ! *1* =
(*ncontents*
(*if proper-symbols-lt 4* (*second ys*) ∧ *numlist-wf vs*
then if (λ*v*. *v* ∈ *set* (*zs-numlist vs*)) ⊨ *zs-formula xs then 1 else 0*
else 0),
1)
proof (*cases proper-symbols-lt 4* (*second ys*))
case *True*
then have *tpsTT* ! *1* = *tpsTTT* ! *1*
using *tpsTT-def* **by** *simp*
then show *?thesis*
using *tpsTTT True* **by** *simp*
next
case *False*
then have *tpsTT* ! *1* = *tpsTT1* ! *1*
using *tpsTT-def* **by** *auto*
then show *?thesis*
using *tpsTT1-def tps1 canrepr-0 False* **by** *auto*
qed

lemma *tmTTI* [*transforms-intros*]:
assumes *ttt* = *140* + *228* ∗ *length zs* ^ *4*
shows *transforms tmTTI tpsTT1 ttt tpsTT*
unfolding *tmTTI-def*
proof (*tform time*: *assms*)
have ∗: *read tpsTT1* ! *10* ≠ □ ⟷ *proper-symbols-lt 4* (*second ys*)
using *tpsTT1-def tps1 read-ncontents-eq-0* **by** *simp*
show *read tpsTT1* ! *10* ≠ □ ⟹ *proper-symbols-lt 4* (*second ys*)
using ∗ **by** *simp*
let *?t* = *138* + *228* ∗ *length zs* ^ *4*
show *read tpsTT1* ! *10* ≠ □ ⟹ *tpsTT* = *tpsTTT*
using ∗ *tpsTT-def* **by** *simp*
show ¬ *read tpsTT1* ! *10* ≠ □ ⟹ *tpsTT* = *tpsTT1*
using ∗ *tpsTT-def* **by** *auto*
qed

lemma *tmTT* [*transforms-intros*]:
assumes *ttt* = *145* + *7* ∗ *length* (*second ys*) + *228* ∗ *length zs* ^ *4*
shows *transforms tmTT tpsT2 ttt tpsTT*
unfolding *tmTT-def* **by** (*tform time*: *assms*)

end

definition *tpsTE* ≡ *tps1*
  [*2* := (⌊*ys*⌋, *1*),
   3 := (⌊*first ys*⌋, *1*),
   4 := (⌊*second ys*⌋, *1*),
   5 := (⌊*even* (*length* (*first ys*))⌋$_B$, *1*),
   6 := (⌊*proper-symbols-lt 4* (*first ys*) ∧ *even* (*length* (*first ys*))⌋$_B$, *1*),
   7 := (⌊*bindecode* (*first ys*)⌋, *1*),
   8 := (⌊*numlistlist-wf xs*⌋$_B$, *1*),
   1 := (⌊*1*⌋$_N$, *1*)]

definition *tpsT* ≡ (*if numlistlist-wf xs then tpsTT else tpsTE*)

lemma *length-tpsT*: *length tpsT* = *22*
  using *tpsT-def length-tpsTT tpsTE-def tps1* **by** *simp*

lemma *tpsT*: *tpsT* ! *1* =
  (*ncontents*
   (*if numlistlist-wf xs*
    then if proper-symbols-lt 4 (*second ys*) ∧ *numlist-wf vs*
         then if (λ*v*. *v* ∈ *set* (*zs-numlist vs*)) ⊨ *zs-formula xs then 1 else 0*
         else 0
    else 1),
   1)
proof (*cases numlistlist-wf xs*)
  case *True*
  then have *tpsT* ! *1* = *tpsTT* ! *1*
    using *tpsT-def* **by** *simp*
  then show *?thesis*
    using *tpsTT True* **by** *simp*
next
  case *False*
  then have *tpsT* ! *1* = *tpsTE* ! *1*
    using *tpsT-def* **by** *auto*
  then show *?thesis*
    using *tpsTE-def tps1 canrepr-0 False* **by** *auto*
qed

lemma *tmTI* [*transforms-intros*]:
  assumes *ttt* = *147* + *7* ∗ *length* (*second ys*) + *228* ∗ *length zs* ^ *4*
  shows *transforms tmTI tpsT2 ttt tpsT*
  unfolding *tmTI-def*
proof (*tform time*: *assms*)
  have ∗: *read tpsT2* ! *8* ≠ □ ⟷ *numlistlist-wf xs*
    using *tpsT2-def tps1 read-ncontents-eq-0* **by** *simp*
  show *read tpsT2* ! *8* ≠ □ ⟹ *numlistlist-wf xs*
    using ∗ **by** *simp*
  show *1* < *length tpsT2*
    using *tpsT2-def tps1* **by** *simp*
  show *tpsT2* ! *1* = (⌊*0*⌋$_N$, *1*)
    using *tpsT2-def tps1 canrepr-0* **by** *simp*
  show ¬ *read tpsT2* ! *8* ≠ □ ⟹ *tpsT* = *tpsT2*[*1* := (⌊*1*⌋$_N$, *1*)]
    using *tpsT-def* ∗ *tpsT2-def tpsTE-def* **by** *presburger*
  show *read tpsT2* ! *8* ≠ □ ⟹ *tpsT* = *tpsTT*
    using ∗ *tpsT-def* **by** *simp*
  show *10* + *2* ∗ *nlength 0* + *2* ∗ *nlength 1* + *1* ≤ *ttt*
    using *assms nlength-1-simp* **by** *simp*
qed

lemma *tmT* [*transforms-intros*]:
  assumes *ttt* = *360* + *3* ∗ *length* (*first ys*) + *39* ∗ *length xs* + *7* ∗ *length* (*second ys*) + *228* ∗ *length zs* ^ *4*
  shows *transforms tmT tps6 ttt tpsT*

```
  unfolding tmT-def by (tform time: assms)

end

definition tpsE ≡ tps1
  [2 := (⌊ys⌋, 1),
   3 := (⌊first ys⌋, 1),
   4 := (⌊second ys⌋, 1),
   5 := (⌊even (length (first ys))⌋_B, 1),
   6 := (⌊proper-symbols-lt 4 (first ys) ∧ even (length (first ys))⌋_B, 1),
   1 := (⌊1⌋_N, 1)]

definition tps7 ≡ (if proper-symbols-lt 4 (first ys) ∧ even (length (first ys)) then tpsT else tpsE)

lemma length-tps7: length tps7 = 22
  using tps7-def length-tpsT tpsE-def tps1 by simp

lemma tps7: tps7 ! 1 =
  (ncontents
   (if proper-symbols-lt 4 (first ys) ∧ even (length (first ys)) ∧ numlistlist-wf xs
    then if proper-symbols-lt 4 (second ys) ∧ numlist-wf vs
         then if (λv. v ∈ set (zs-numlist vs)) ⊨ zs-formula xs then 1 else 0
         else 0
    else 1),
   1)
proof (cases proper-symbols-lt 4 (first ys) ∧ even (length (first ys)))
  case True
  then have tps7 ! 1 = tpsT ! 1
    using tps7-def by simp
  then show ?thesis
    using tpsT True by simp
next
  case False
  then have tps7 ! 1 = tpsE ! 1
    using tps7-def by auto
  then show ?thesis
    using tpsE-def tps1 canrepr-0 False by auto
qed

lemma tps7': tps7 ! 1 = (⌊verify-sat zs⌋, 1)
proof −
  have proper-symbols-lt 4 zs = bit-symbols zs for zs
    by fastforce
  then show ?thesis
    unfolding verify-sat-def Let-def using tps7 canrepr-0 canrepr-1 by auto
qed

lemma tmI [transforms-intros]:
  assumes ttt = 362 + 3 ∗ length (first ys) + 39 ∗ length xs + 7 ∗ length (second ys) + 228 ∗ length zs ^ 4
  shows transforms tmI tps6 ttt tps7
  unfolding tmI-def
proof (tform time: assms)
  have ∗: read tps6 ! 6 ≠ □ ⟷ (proper-symbols-lt 4 (first ys)) ∧ even (length (first ys))
    using tps6-def tps1 read-ncontents-eq-0 by simp
  show read tps6 ! 6 ≠ □ ⟹ (proper-symbols-lt 4 (first ys)) ∧ even (length (first ys))
    using ∗ by simp
  show 1 < length tps6
    using tps6-def tps1 by simp
  show tps6 ! 1 = (⌊0⌋_N, 1)
    using tps6-def tps1 canrepr-0 by simp
  show ¬ read tps6 ! 6 ≠ □ ⟹ tps7 = tps6[1 := (⌊1⌋_N, 1)]
    using tps7-def ∗ tps6-def tpsE-def by metis
  show read tps6 ! 6 ≠ □ ⟹ tps7 = tpsT
```

using *tps7-def* ∗ **by** *simp*
show *10 + 2 ∗ nlength 0 + 2 ∗ nlength 1 + 1 ≤ ttt*
using *assms nlength-1-simp* **by** *simp*
qed

lemma *tm7*:
assumes *ttt = 398 + 3 ∗ length zs + 6 ∗ length ys + 17 ∗ length (first ys) +
39 ∗ length xs + 7 ∗ length (second ys) + 228 ∗ length zs ^ 4*
shows *transforms tm7 tps0 ttt tps7*
unfolding *tm7-def* **by** (*tform time*: *assms*)

lemma *tm7′* [*transforms-intros*]:
assumes *ttt = 398 + 300 ∗ length zs ^ 4*
shows *transforms tm7 tps0 ttt tps7*
proof −
have ∗: *length ys ≤ length zs*
by *simp*
then have *1*: *length (second ys) ≤ length zs*
using *length-second dual-order.trans* **by** *blast*
have *2*: *length (first ys) ≤ length zs*
using ∗ *dual-order.trans length-first* **by** *blast*
then have *3*: *length xs ≤ length zs*
by *simp*
let *?ttt = 398 + 3 ∗ length zs + 6 ∗ length ys + 17 ∗ length (first ys) +
39 ∗ length xs + 7 ∗ length (second ys) + 228 ∗ length zs ^ 4*
have *?ttt ≤ 398 + 9 ∗ length zs + 17 ∗ length zs + 39 ∗ length zs + 7 ∗ length zs + 228 ∗ length zs ^ 4*
using ∗ *1 2 3* **by** *simp*
also have ... *= 398 + 72 ∗ length zs + 228 ∗ length zs ^ 4*
by *simp*
also have ... *≤ 398 + 72 ∗ length zs ^ 4 + 228 ∗ length zs ^ 4*
using *linear-le-pow* **by** *simp*
also have ... *= 398 + 300 ∗ length zs ^ 4*
by *simp*
finally have *?ttt ≤ 398 + 300 ∗ length zs ^ 4* **.**
then show *?thesis*
using *assms tm7 transforms-monotone* **by** *fast*
qed

end

end

lemma *transforms-tm-verify-sat*:
fixes *zs* :: *symbol list* **and** *tps* :: *tape list*
assumes *bit-symbols zs*
and *tps = snd (start-config 22 zs)*
and *ttt = 398 + 300 ∗ length zs ^ 4*
shows ∃ *tps′. tps′ ! 1 = (⌊verify-sat zs⌋, 1) ∧ transforms tm-verify-sat tps ttt tps′*
proof −
interpret *loc*: *turing-machine-verify-sat* **.**
show *?thesis*
using *assms loc.tm7′ loc.tps7′ loc.tm7-eq-tm-verify-sat* **by** *metis*
qed

With the Turing machine just constructed and the polynomial $p(n) = n$ we can satisfy the definition of $\mathcal{NP}$ and prove the main result of this chapter.

theorem *SAT-in-NP*: *SAT* ∈ $\mathcal{NP}$
proof −
define *p* :: *nat* ⇒ *nat* **where** *p* = (λ*n*. *n*)
define *T* :: *nat* ⇒ *nat* **where** *T* = (λ*n*. *398 + 300 ∗ n ^ 4*)
define *f* :: *string* ⇒ *string* **where**
f = (λ*x*. *symbols-to-string* (*verify-sat* (*string-to-symbols x*)))
have *turing-machine 22 6 tm-verify-sat*

  using *tm-verify-sat-tm* **.**
moreover have *polynomial p*
  using *p-def polynomial-id* **by** (*metis eq-id-iff*)
moreover have *big-oh-poly T*
  using *T-def big-oh-poly-poly big-oh-poly-const big-oh-poly-sum big-oh-poly-prod* **by** *simp*
moreover have *computes-in-time 22 tm-verify-sat f T*
proof
  fix *x* :: *string*
  let *?zs* = *string-to-symbols x*
  have *bs*: *bit-symbols ?zs*
    by *simp*
  have *bit-symbols* (*verify-sat ?zs*)
    using *bit-symbols-verify-sat* **by** *simp*
  then have ∗: *string-to-symbols* (*f x*) = *verify-sat ?zs*
    unfolding *f-def* **using** *bit-symbols-to-symbols* **by** *simp*

  obtain *tps* **where** *tps*:
      tps ! *1* = (⌊*verify-sat ?zs*⌋, *1*)
      transforms tm-verify-sat (*snd* (*start-config 22 ?zs*)) (*T* (*length ?zs*)) *tps*
    using *bs transforms-tm-verify-sat T-def* **by** *blast*
  then have *tps* ::: *1* = *string-to-contents* (*f x*)
    using ∗ *start-config-def contents-string-to-contents* **by** *simp*
  then show ∃ *tps. tps* ::: *1* = *string-to-contents* (*f x*) ∧
      transforms tm-verify-sat (*snd* (*start-config-string 22 x*)) (*T* (*length x*)) *tps*
    using *tps*(*2*) **by** *auto*
qed
moreover have ∀ *x*. *x* ∈ *SAT* ⟷ (∃ *u*. *length u* = *p* (*length x*) ∧ *f* ⟨*x*, *u*⟩ = [𝕀])
proof
  fix *x* :: *string*
  show (*x* ∈ *SAT*) = (∃ *u*. *length u* = *p* (*length x*) ∧ *f* ⟨*x*, *u*⟩ = [𝕀])
  proof
    show ∃ *u*. *length u* = *p* (*length x*) ∧ *f* ⟨*x*, *u*⟩ = [𝕀] **if** *x* ∈ *SAT*
    proof (*cases* ∃ φ. *x* = *formula-to-string* φ)
      case *True*
      then obtain φ **where** φ: *x* = *formula-to-string* φ *satisfiable* φ
        using *SAT-def* **using** ‹*x* ∈ *SAT*› **by** *auto*
      then obtain *us* **where** *us*:
          bit-symbols us
          length us = *length* (*formula-to-string* φ)
          verify-sat ⟨*formula-to-string* φ; *symbols-to-string us*⟩ = [**1**]
        using *ex-witness-linear-length* **by** *blast*
      let *?zs* = ⟨*formula-to-string* φ; *symbols-to-string us*⟩
      define *u* **where** *u* = *symbols-to-string us*
      have *length us* = *p* (*length x*)
        using *us*(*2*) φ(*1*) *p-def* **by** *simp*
      then have *1*: *length u* = *p* (*length x*)
        using *u-def* **by** *simp*

      have *f* ⟨*x*, *u*⟩ = *symbols-to-string* (*verify-sat* ⟨*x*; *u*⟩)
        using *f-def* **by** *simp*
      also have ... = *symbols-to-string* (*verify-sat ?zs*)
        using φ(*1*) *u-def* **by** *simp*
      also have ... = *symbols-to-string* [**1**]
        using *us*(*3*) **by** *simp*
      also have ... = [𝕀]
        by *simp*
      finally have *f* ⟨*x*, *u*⟩ = [𝕀] **.**
      then show *?thesis*
        using *1* **by** *auto*
    next
      case *False*
      define *u* **where** *u* = *replicate* (*length x*) 𝕆
      then have *1*: *length u* = *p* (*length x*)

using *p-def* **by** *simp*
have $f \langle x, u\rangle = $ *symbols-to-string* (*verify-sat* $\langle x; u\rangle$)
using *f-def* **by** *simp*
also have ... = *symbols-to-string* [**1**]
using *verify-sat-not-wf-phi False* **by** *simp*
also have ... = [𝕀]
by *simp*
finally have $f \langle x, u\rangle = $ [𝕀] **.**
then show *?thesis*
using *1* **by** *auto*
qed
show $x \in SAT$ **if** *ex*: $\exists u.$ *length* $u = p$ (*length* x) $\wedge$ $f \langle x, u\rangle = $ [𝕀]
proof (*rule ccontr*)
assume *notin*: $x \notin SAT$
then obtain φ **where** φ: $x = $ *formula-to-string* φ $\neg$ *satisfiable* φ
using *SAT-def* **by** *auto*
obtain u **where** u: *length* $u = p$ (*length* x) $f \langle x, u\rangle = $ [𝕀]
using *ex* **by** *auto*
have $f \langle x, u\rangle = $ *symbols-to-string* (*verify-sat* $\langle x; u\rangle$)
using *f-def* **by** *simp*
also have ... = *symbols-to-string* (*verify-sat* $\langle$*formula-to-string* φ; $u\rangle$)
using φ(*1*) **by** *simp*
also have ... = *symbols-to-string* []
using *verify-sat-not-sat* φ(*2*) **by** *simp*
also have ... = []
by *simp*
finally have $f \langle x, u\rangle = $ [] **.**
then show *False*
using u(*2*) **by** *simp*
qed
qed
qed
ultimately show *?thesis*
using *complexity-class-NP-def* **by** *fast*
qed

end

Chapter 5

Obliviousness

In order to show that `SAT` is $\mathcal{NP}$-hard we will eventually show how to reduce an arbitrary language $L \in \mathcal{NP}$ to `SAT`. The proof can only use properties of L common to all languages in $\mathcal{NP}$. The definition of $\mathcal{NP}$ provides us with a verifier Turing machine M for L, of which we only know that it is running in polynomial time. In addition by lemma *NP-output-len-1* we can assume that M outputs a single bit symbol. In this chapter we are going to show that we can make additional assumptions about M, namely:

1. M has only two tapes.
2. M halts on $\langle x, u\rangle$ with the output tape head on the symbol $\mathbf{1}$ iff. u is a certificate for x.
3. M is *oblivious*, which means that on any input x the head positions of M on all its tapes depend only on the *length* of x, not on the symbols in x [2, Remark 1.7].

These additional properties will somewhat simplify the reduction of L to `SAT`, more precisely the construction of the CNF formulas (see Chapter 6).

In order to achieve this goal we will show how to simulate any polynomial-time multi-tape TM in polynomial time on a two-tape oblivious TM that halts with the output tape head on cell 1.

Given a polynomial-time k-tape TM M, the basic approach is to construct a two-tape TM that encodes the k tapes of M on its output tape in such a way that every cell encodes k symbols of M and flags for M's tape heads. This is the same idea as used by Dalvit and Thiemann [5] and originally Hartmanis and Stearns [9] for simulating a multi-tape TM on a single-tape TM. After all our two-tape simulator can only properly use a single tape (the output/work tape). This simulator has roughly a quadratic running time overhead and so keeps the running time polynomial. However, it is not generally an oblivious TM.

To make the simulator TM oblivious, we have it initially "format" a section on the output tape that is long enough to hold everything M is going to write and whose length only depends on the input length. To simulate one step of M, the simulator then sweeps its output tape head all the way from the start of the tape to the end of the formatted space and back again, moving one cell per step. During these sweeps it executes one step of the simulation of M. Since the size of the formatted space only depends on the input length, the simulator performs the same head movements on inputs of the same length, resulting in an oblivious behavior. Moreover, it is easy to make it halt with the output tape head on cell number 1.

The formatter TM is described in Section 5.2. The simulator TM is then constructed in Section 5.3. Finally Section 5.4 states the main result of this chapter.

Before any of this, however, we have to define some basic concepts surrounding obliviousness.

5.1 Oblivious Turing machines

theory *Oblivious*
 imports *Memorizing*
begin

This section provides us with the tools for showing that a Turing machine is oblivious and for combining oblivious TMs into more complex oblivious TMs.

So far our analysis of Turing machines involved their semantics and running time bounds. For this we mainly used the *transforms* predicate, which relates a start configuration and a halting configuration and an upper bound for the running time of a TM to transit from the one configuration to the other. To deal with obliviousness, we need to look more closely and inspect the sequence of tape head positions during the TM's execution, rather than only the running time.

The subsections in this section roughly correspond to Sections 2.1 to 2.5. In the first subsection we introduce a predicate *trace* analogous to *transforms* and show its behavior under sequential composition of TMs and loops (we will not need branches). The next subsection shows the head position sequences for those few elementary TMs from Section 2.4 that we need for our more complex oblivious TMs later. These constructions will also heavily use the memorization-in-states technique from Section 2.5, which we adapt to this chapter's needs in the final subsection.

5.1.1 Traces and head positions

In order to show that a Turing machine is oblivious we need to keep track of its head positions. Consider a machine M that transits from a configuration *cfg1* to a configuration *cfg2* in t steps. We call the sequence of head positions on the first two tapes a *trace*. If we ignore the initial head positions, the length of a trace equals t. Moreover we will only consider traces where M either does not halt or halts in the very last step. These two properties mean, for example, that we can simply concatenate a trace of a TM that halts and trace of another TM and get the trace of the sequential execution of both TMs. Similarly, analysing while loops is simplified by these two extra assumptions. The next predicate defines what it means for a list *es* to be a trace.

```
definition trace :: machine ⇒ config ⇒ (nat × nat) list ⇒ config ⇒ bool where
  trace M cfg1 es cfg2 ≡
    execute M cfg1 (length es) = cfg2 ∧
    (∀ i<length es. fst (execute M cfg1 i) < length M) ∧
    (∀ i<length es. execute M cfg1 (Suc i) <#> 0 = fst (es ! i)) ∧
    (∀ i<length es. execute M cfg1 (Suc i) <#> 1 = snd (es ! i))
```

We will consider traces for machines with more than two tapes, too, but only for auxiliary constructions in combination with the memorizing-in-states technique. Therefore our definition is limited to start configurations with two tapes. A machine is *oblivious* if there is a function mapping the input length to the trace that takes the machine from the start configuration with that input to a halting configuration.

```
definition oblivious :: machine ⇒ bool where
  oblivious M ≡ ∃ e.
   (∀ zs. bit-symbols zs ⟶ (∃ tps. trace M (start-config 2 zs) (e (length zs)) (length M, tps)))

lemma trace-Nil: trace M cfg [] cfg
  unfolding trace-def by simp

lemma traceI:
  assumes execute M (q1, tps1) (length es) = (q2, tps2)
    and ⋀i. i < length es ⟹ fst (execute M (q1, tps1) i) < length M
    and ⋀i. i < length es ⟹
      execute M (q1, tps1) (Suc i) <#> 0 = fst (es ! i) ∧
      execute M (q1, tps1) (Suc i) <#> 1 = snd (es ! i)
  shows trace M (q1, tps1) es (q2, tps2)
  using trace-def assms by simp

lemma traceI':
  assumes execute M cfg1 (length es) = cfg2
    and ⋀i. i < length es ⟹ fst (execute M cfg1 i) < length M
    and ⋀i. i < length es ⟹
      execute M cfg1 (Suc i) <#> 0 = fst (es ! i) ∧
      execute M cfg1 (Suc i) <#> 1 = snd (es ! i)
  shows trace M cfg1 es cfg2
  using trace-def assms by simp

lemma trace-additive:
  assumes trace M (q1, tps1) es1 (q2, tps2) and trace M (q2, tps2) es2 (q3, tps3)
  shows trace M (q1, tps1) (es1 @ es2) (q3, tps3)
proof (rule traceI)
```

```
  let ?es = es1 @ es2
  show execute M (q1, tps1) (length (es1 @ es2)) = (q3, tps3)
    using trace-def assms by (simp add: execute-additive)
  show fst (execute M (q1, tps1) i) < length M if i < length ?es for i
  proof (cases i < length es1)
    case True
    then show ?thesis
      using that assms(1) trace-def by simp
  next
    case False
    have execute M (q1, tps1) (length es1 + (i − length es1)) = execute M (q2, tps2) (i − length es1)
      using execute-additive that assms(1) trace-def by blast
    then have ∗: execute M (q1, tps1) i = execute M (q2, tps2) (i − length es1)
      using False by simp
    have i − length es1 < length es2
      using that False by simp
    then have fst (execute M (q2, tps2) (i − length es1)) < length M
      using assms(2) trace-def by simp
    then show ?thesis
      using ∗ by simp
  qed
  show execute M (q1, tps1) (Suc i) <#> 0 = fst (?es ! i) ∧
        execute M (q1, tps1) (Suc i) <#> 1 = snd (?es ! i)
    if i < length ?es for i
  proof (cases i < length es1)
    case True
    then show ?thesis
      using that assms(1) trace-def by (simp add: nth-append)
  next
    case False
    have execute M (q1, tps1) (length es1 + (Suc i − length es1)) = execute M (q2, tps2) (Suc i − length es1)
      using execute-additive that assms(1) trace-def by blast
    then have ∗: execute M (q1, tps1) (Suc i) = execute M (q2, tps2) (Suc (i − length es1))
      using False by (simp add: Suc-diff-le)
    have i − length es1 < length es2
      using that False by simp
    then have execute M (q2, tps2) (Suc (i − length es1)) <#> 0 = fst (es2 ! (i − length es1))
      and execute M (q2, tps2) (Suc (i − length es1)) <#> 1 = snd (es2 ! (i − length es1))
      using assms(2) trace-def by simp-all
    then show ?thesis
      using ∗ by (simp add: False nth-append)
  qed
qed

lemma trace-additive′:
  assumes trace M cfg1 es1 cfg2 and trace M cfg2 es2 cfg3
  shows trace M cfg1 (es1 @ es2) cfg3
  using trace-additive assms by (metis prod.collapse)
```

We mostly consider traces from the start state to the halting state, for which we introduce the next predicate.

```
definition traces :: machine ⇒ tape list ⇒ (nat × nat) list ⇒ tape list ⇒ bool where
  traces M tps1 es tps2 ≡ trace M (0, tps1) es (length M, tps2)
```

The relation between *traces* and *trace* is like that between *transforms* and *transits*.

```
lemma tracesI [intro]:
  assumes execute M (0, tps1) (length es) = (length M, tps2)
    and ⋀i. i < length es ⟹ fst (execute M (0, tps1) i) < length M
    and ⋀i. i < length es ⟹
      execute M (0, tps1) (Suc i) <#> 0 = fst (es ! i) ∧
      execute M (0, tps1) (Suc i) <#> 1 = snd (es ! i)
  shows traces M tps1 es tps2
  using traces-def trace-def assms by simp
```

lemma *traces-additive*:
  assumes *trace M* (*0*, *tps1*) *es1* (*0*, *tps2*)
    and *traces M tps2 es2 tps3*
  shows *traces M tps1* (*es1* @ *es2*) *tps3*
  using *assms traces-def trace-additive* **by** *simp*

lemma *execute-trace-append*:
  assumes *trace M1* (*0*, *tps1*) *es1* (*length M1*, *tps2*) (**is** *trace* - *?cfg1* - -)
    and $t \leq$ *length es1*
  shows *execute* (*M1* @ *M2*) (*0*, *tps1*) *t* = *execute M1* (*0*, *tps1*) *t*
    (**is** *execute ?M* - - = -)
  using *assms*(*2*)
proof (*induction t*)
  case *0*
  then show *?case*
    by *simp*
next
  case (*Suc t*)
  then have *t* < *length es1*
    by *simp*
  then have *1*: *fst* (*execute M1 ?cfg1 t*) < *length M1*
    using *traces-def trace-def assms*(*1*) **by** *simp*
  have *2*: *length ?M* = *length M1* + *length M2*
    using *length-turing-machine-sequential* **by** *simp*
  have *execute ?M ?cfg1* (*Suc t*) = *exe ?M* (*execute ?M ?cfg1 t*)
    by *simp*
  also have ... = *exe ?M* (*execute M1 ?cfg1 t*) (**is** - = *exe* - *?cfg*)
    using *Suc* **by** *simp*
  also have ... = *sem* (*?M* ! (*fst ?cfg*)) *?cfg*
    using *1 2 exe-def* **by** *simp*
  also have ... = *sem* (*M1* ! (*fst ?cfg*)) *?cfg*
    using *1* **by** (*simp add*: *nth-append turing-machine-sequential-def*)
  also have ... = *exe M1* (*execute M1 ?cfg1 t*)
    using *exe-def 1* **by** *simp*
  also have ... = *execute M1 ?cfg1* (*Suc t*)
    by *simp*
  finally show *?case* **.**
qed

5.1.2 Increasing the number of tapes

This is lemma *transforms-append-tapes* adapted for *traces*.

lemma *traces-append-tapes*:
  assumes *turing-machine 2 G M* **and** *length tps1* = *2* **and** *traces M tps1 es tps2*
  shows *traces* (*append-tapes 2* (*2* + *length tps'*) *M*) (*tps1* @ *tps'*) *es* (*tps2* @ *tps'*)
proof
  let *?M* = *append-tapes 2* (*2* + *length tps'*) *M*
  show *execute ?M* (*0*, *tps1* @ *tps'*) (*length es*) = (*length ?M*, *tps2* @ *tps'*)
  proof −
    have *execute M* (*0*, *tps1*) (*length es*) = (*length M*, *tps2*)
      using *assms*(*3*) **by** (*simp add*: *trace-def traces-def*)
    moreover have *execute ?M* (*0*, *tps1* @ *tps'*) (*length es*) =
        (*fst* (*execute M* (*0*, *tps1*) (*length es*)), *snd* (*execute M* (*0*, *tps1*) (*length es*)) @ *tps'*)
      using *execute-append-tapes'*[*OF assms*(*1*−*2*)] **by** *simp*
    ultimately show *?thesis*
      by (*simp add*: *length-append-tapes*)
  qed
  show *fst* (*execute ?M* (*0*, *tps1* @ *tps'*) *i*) < *length ?M* **if** *i* < *length es* **for** *i*
  proof −
    have *fst* (*execute M* (*0*, *tps1*) *i*) < *length M*
      using *that assms*(*3*) *trace-def traces-def* **by** *blast*
    then show *fst* (*execute ?M* (*0*, *tps1* @ *tps'*) *i*) < *length ?M*

```
    by (metis (no-types) assms(1,2) execute-append-tapes' fst-conv length-append-tapes)
  qed
  show snd (execute ?M (0, tps1 @ tps') (Suc i)) :#: 0 = fst (es ! i) ∧
       snd (execute ?M (0, tps1 @ tps') (Suc i)) :#: 1 = snd (es ! i)
    if i < length es for i
  proof -
    have snd (execute ?M (0, tps1 @ tps') (Suc i)) = snd (execute M (0, tps1) (Suc i)) @ tps'
      using execute-append-tapes' assms by (metis snd-conv)
    moreover have ||execute M (0, tps1) (Suc i)|| = 2
      using assms(1,2) by (metis execute-num-tapes snd-conv)
    ultimately show ?thesis
      using that assms by (simp add: nth-append trace-def traces-def)
  qed
qed
```

5.1.3 Combining Turing machines

Traces for sequentially composed Turing machines are just concatenated traces of the individual machines.

```
lemma traces-sequential:
  assumes traces M1 tps1 es1 tps2 and traces M2 tps2 es2 tps3
  shows traces (M1 ;; M2) tps1 (es1 @ es2) tps3
proof
  let ?M = M1 ;; M2
  let ?cfg1 = (0, tps1)
  let ?cfg1' = (length M1, tps2)
  let ?cfg2 = (0, tps2)
  let ?cfg2' = (length M2, tps3)
  let ?es = es1 @ es2
  have 3: execute M1 ?cfg1 (length es1) = ?cfg1'
    using assms(1) traces-def trace-def by simp
  have fst ?cfg1 = 0
    by simp
  have 4: execute M2 ?cfg2 (length es2) = ?cfg2'
    using assms(2) traces-def trace-def by auto
  have ?cfg1' = ?cfg2 <+=> length M1
    by simp
  have 2: length ?M = length M1 + length M2
    using length-turing-machine-sequential by simp
  have t-le: execute ?M ?cfg1 t = execute M1 ?cfg1 t if t ≤ length es1 for t
    using that
  proof (induction t)
    case 0
    then show ?case
      by simp
  next
    case (Suc t)
    then have t < length es1
      by simp
    then have 1: fst (execute M1 ?cfg1 t) < length M1
      using traces-def trace-def assms(1) by simp
    have execute ?M ?cfg1 (Suc t) = exe ?M (execute ?M ?cfg1 t)
      by simp
    also have ... = exe ?M (execute M1 ?cfg1 t) (is - = exe - ?cfg)
      using Suc by simp
    also have ... = sem (?M ! (fst ?cfg)) ?cfg
      using 1 2 exe-def by simp
    also have ... = sem (M1 ! (fst ?cfg)) ?cfg
      using 1 by (simp add: nth-append turing-machine-sequential-def)
    also have ... = exe M1 (execute M1 ?cfg1 t)
      using exe-def 1 by simp
    also have ... = execute M1 ?cfg1 (Suc t)
      by simp
    finally show ?case .
```

```
qed
have t-ge: execute ?M ?cfg1 (length es1 + t) = execute M2 ?cfg2 t <+=> length M1
    if t ≤ length es2 for t
  using that
proof (induction t)
  case 0
  then show ?case
    using t-le 3 by simp
next
  case (Suc t)
  have execute ?M ?cfg1 (length es1 + Suc t) = execute ?M ?cfg1 (Suc (length es1 + t))
    by simp
  also have ... = exe ?M (execute ?M ?cfg1 (length es1 + t))
    by simp
  also have ... = exe ?M (execute M2 ?cfg2 t <+=> length M1)
      (is - = exe - (?cfg <+=> -))
    using Suc by simp
  also have ... = (exe M2 (execute M2 ?cfg2 t)) <+=> length M1
    using exe-relocate by simp
  also have ... = execute M2 ?cfg2 (Suc t) <+=> length M1
    by simp
  finally show ?case .
qed
show fst (execute ?M ?cfg1 i) < length ?M if i < length ?es for i
proof (cases i < length es1)
  case True
  then show ?thesis
    using t-le assms(1) traces-def trace-def 2 by auto
next
  case False
  then obtain i' where i = length es1 + i' i' ≤ length es2
    by (metis ‹i < length (es1 @ es2)› add-diff-inverse-nat add-le-cancel-left length-append less-or-eq-imp-le)
  then show ?thesis
    using t-ge assms(2) traces-def trace-def that 2 by simp
qed
show execute ?M ?cfg1 (length ?es) = (length ?M, tps3)
  by (simp add: 2 4 t-ge)
show execute ?M ?cfg1 (Suc i) <#> 0 = fst (?es ! i) ∧
      execute ?M ?cfg1 (Suc i) <#> 1 = snd (?es ! i)
      if i < length ?es for i
proof (cases i < length es1)
  case True
  then have Suc i ≤ length es1
    by simp
  then have execute ?M ?cfg1 (Suc i) = execute M1 ?cfg1 (Suc i)
    using t-le by blast
  then show ?thesis
    using assms(1) traces-def trace-def by (simp add: True nth-append)
next
  case False
  have 8: i − length es1 < length es2
    using False that by simp
  with False have Suc i − length es1 ≤ length es2
    by simp
  then have execute ?M ?cfg1 (Suc i) = execute M2 ?cfg2 (Suc i − length es1) <+=> length M1
    using t-ge False by fastforce
  moreover have ?es ! i = es2 ! (i − length es1)
    by (simp add: False nth-append)
  moreover have execute M2 ?cfg2 (Suc i) <#> 0 = fst (es2 ! i) ∧
      execute M2 ?cfg2 (Suc i) <#> 1 = snd (es2 ! i) if i < length es2 for i
      using that assms(2) traces-def trace-def by simp
  ultimately show ?thesis
    by (metis 8 False Nat.add-diff-assoc le-less-linear plus-1-eq-Suc snd-conv)
```

```
  qed
qed
```

Next we show how to derive traces for machines created by the *WHILE* operation. If the condition is false, the trace of the loop is the trace for the machine computing the condition plus a singleton trace for the jump.

```
lemma tm-loop-sem-false-trace:
  assumes traces M1 tps0 es1 tps1
    and ¬ cond (read tps1)
  shows trace
          (WHILE M1 ; cond DO M2 DONE)
          (0, tps0)
          (es1 @ [(tps1 :#: 0, tps1 :#: 1)])
          (length M1 + length M2 + 2, tps1)
    (is trace ?M - - -)
proof (rule traceI)
  let ?C1 = M1
  let ?C2 = [cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)]
  let ?C3 = relocate (length M1 + 1) M2
  let ?C4 = [cmd-jump (λ-. True) 0 0]
  let ?C34 = ?C3 @ ?C4
  have parts: ?M = ?C1 @ ?C2 @ ?C3 @ ?C4
    using turing-machine-loop-def by simp
  then have 1: ?M ! (length M1) = cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)
    by simp
  let ?es = es1 @ [(tps1 :#: 0, tps1 :#: 1)]
  show goal1: execute ?M (0, tps0) (length ?es) = (length M1 + length M2 + 2, tps1)
  proof −
    have execute ?M (0, tps0) (length es1) = execute M1 (0, tps0) (length es1)
      using execute-trace-append assms by (simp add: traces-def turing-machine-loop-def)
    then have 2: execute ?M (0, tps0) (length es1) = (length M1, tps1)
      using assms trace-def traces-def by simp
    have execute ?M (0, tps0) (length ?es) = execute ?M (0, tps0) (Suc (length es1))
      by simp
    also have ... = exe ?M (execute ?M (0, tps0) (length es1))
      by simp
    also have ... = exe ?M (length M1, tps1)
      using 2 by simp
    also have ... = sem (cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)) (length M1, tps1)
      by (simp add: 1 exe-lt-length turing-machine-loop-len)
    also have ... = (length M1 + length M2 + 2, tps1)
      using assms(2) sem-jump by simp
    finally show ?thesis .
  qed
  show fst (execute ?M (0, tps0) i) < length ?M if i < length ?es for i
  proof (cases i < length es1)
    case True
    then have execute ?M (0, tps0) i = execute M1 (0, tps0) i
      using execute-trace-append assms parts by (simp add: traces-def)
    then show ?thesis
      using assms(1) trace-def traces-def True turing-machine-loop-len by auto
  next
    case False
    with that have i = length es1
      by simp
    then show ?thesis
      using assms(1) trace-def traces-def turing-machine-loop-len
      by (simp add: execute-trace-append parts)
  qed
  show execute ?M (0, tps0) (Suc i) <#> 0 = fst (?es ! i) ∧
      execute ?M (0, tps0) (Suc i) <#> 1 = snd (?es ! i)
    if i < length ?es for i
  proof (cases i < length es1)
```

```
    case True
    then have Suc i ≤ length es1
      by simp
    then have execute ?M (0, tps0) (Suc i) = execute M1 (0, tps0) (Suc i)
      using execute-trace-append assms parts by (metis traces-def)
    then show ?thesis
      using assms(1) trace-def traces-def True by (simp add: nth-append)
  next
    case False
    with that have Suc i = length ?es
      by simp
    then show ?thesis
      using goal1 by simp
  qed
qed

lemma tm-loop-sem-false-traces:
  assumes traces M1 tps0 es1 tps1
    and ¬ cond (read tps1)
    and es = es1 @ [(tps1 :#: 0, tps1 :#: 1)]
  shows traces (WHILE M1 ; cond DO M2 DONE) tps0 es tps1
  using tm-loop-sem-false-trace assms traces-def turing-machine-loop-len by fastforce
```

If the loop condition evaluates to true, the trace of one iteration is the concatenation of the traces of the condition machine and the loop body machine with two additional singleton traces for the jumps.

```
lemma tm-loop-sem-true-traces:
  assumes traces M1 tps0 es1 tps1
    and traces M2 tps1 es2 tps2
    and cond (read tps1)
  shows trace
          (WHILE M1 ; cond DO M2 DONE)
          (0, tps0)
          (es1 @ [(tps1 :#: 0, tps1 :#: 1)] @ es2 @ [(tps2 :#: 0, tps2 :#: 1)])
          (0, tps2)
    (is trace ?M - ?es -)
proof (rule traceI)
  let ?C1 = M1
  let ?C2 = [cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)]
  let ?C3 = relocate (length M1 + 1) M2
  let ?C4 = [cmd-jump (λ-. True) 0 0]
  let ?C34 = ?C3 @ ?C4
  have parts: ?M = ?C1 @ ?C2 @ ?C3 @ ?C4
    using turing-machine-loop-def by simp
  then have 1: ?M ! (length M1) = cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)
    by simp
  from parts have parts': ?M = ((?C1 @ ?C2) @ ?C3) @ ?C4
    by simp
  have len-M: length ?M = length M1 + length M2 + 2
    using turing-machine-loop-len assms by simp
  have len-es: length ?es = length es1 + length es2 + 2
    by simp

  have exec-1: execute ?M (0, tps0) t = execute M1 (0, tps0) t if t ≤ length es1 for t
    using execute-trace-append assms by (simp add: parts that traces-def)

  have exec-2: execute ?M (0, tps0) (length es1 + 1) = (length M1 + 1, tps1)
  proof -
    have execute ?M (0, tps0) (length es1) = execute M1 (0, tps0) (length es1)
      using execute-trace-append assms by (simp add: traces-def turing-machine-loop-def)
    then have 2: execute ?M (0, tps0) (length es1) = (length M1, tps1)
      using assms trace-def traces-def by simp
    have execute ?M (0, tps0) (length es1 + 1) = execute ?M (0, tps0) (Suc (length es1))
      by simp
```

```
  also have ... = exe ?M (execute ?M (0, tps0) (length es1))
    by simp
  also have ... = exe ?M (length M1, tps1)
    using 2 by simp
  also have ... = sem (cmd-jump cond (length M1 + 1) (length M1 + length M2 + 2)) (length M1, tps1)
    by (simp add: 1 exe-lt-length turing-machine-loop-len)
  also have ... = (length M1 + 1, tps1)
    using assms(3) sem-jump by simp
  finally show ?thesis .
qed

have exec-3': execute ?M (0, tps0) (length es1 + 1 + t) = execute M2 (0, tps1) t <+=> (length M1 + 1)
  if t ≤ length es2 for t
  using that
proof (induction t)
  case 0
  then show ?case
    using exec-2 by simp
next
  case (Suc t)
  then have 2: fst (execute M2 (0, tps1) t) < length M2
    using assms(2) trace-def traces-def by simp
  then have 3: fst (execute M2 (0, tps1) t <+=> (length M1 + 1)) < length M1 + length M2 + 1
    by simp
  have 4: fst (execute M2 (0, tps1) t <+=> (length M1 + 1)) ≥ length M1 + 1
    by simp

  have ?M = (?C1 @ ?C2) @ (?C3 @ ?C4)
    using parts by simp
  moreover have length (?C1 @ ?C2) = length M1 + 1
    by simp
  ultimately have ?M ! i = (?C3 @ ?C4) ! (i − (length M1 + 1))
      if i ≥ length M1 + 1 and i < length M1 + length M2 + 1 for i
    using that by (simp add: nth-append)
  then have ?M ! i = ?C3 ! (i − (length M1 + 1))
      if i ≥ length M1 + 1 and i < length M1 + length M2 + 1 for i
    using that by (simp add: length-relocate less-diff-conv2 nth-append)
  with 3 4 have ?M ! (fst (execute M2 (0, tps1) t <+=> (length M1 + 1))) =
      ?C3 ! ((fst (execute M2 (0, tps1) t <+=> (length M1 + 1))) − (length M1 + 1))
    by simp
  then have in-C3: ?M ! (fst (execute M2 (0, tps1) t <+=> (length M1 + 1))) =
      ?C3 ! ((fst (execute M2 (0, tps1) t)))
    by simp

  have execute ?M (0, tps0) (length es1 + 1 + Suc t) = execute ?M (0, tps0) (Suc (length es1 + 1 + t))
    by simp
  also have ... = exe ?M (execute ?M (0, tps0) (length es1 + 1 + t))
    by simp
  also have ... = exe ?M (execute M2 (0, tps1) t <+=> (length M1 + 1))
      (is - = exe ?M ?cfg)
    using Suc by simp
  also have ... = sem (?M ! (fst ?cfg)) ?cfg
    using exe-def 3 len-M by simp
  also have ... = sem (?C3 ! (fst (execute M2 (0, tps1) t))) (execute M2 (0, tps1) t)
    using in-C3 sem by simp
  also have ... = sem (M2 ! (fst (execute M2 (0, tps1) t))) (execute M2 (0, tps1) t) <+=> (length M1 + 1)
    using sem-relocate 2 by simp
  also have ... = exe M2 (execute M2 (0, tps1) t) <+=> (length M1 + 1)
    by (simp add: 2 exe-def)
  also have ... = (execute M2 (0, tps1) (Suc t)) <+=> (length M1 + 1)
    by simp
  finally show ?case .
qed
```

```
  then have exec-3: execute ?M (0, tps0) t = execute M2 (0, tps1) (t − (length es1 + 1)) <+=> (length M1
+ 1)
      if t ≥ length es1 + 1 and t ≤ length es1 + length es2 + 1 for t
    using that
      by (smt (verit) Nat.add-diff-assoc2 Nat.diff-diff-right add-diff-cancel-left′ add-diff-cancel-right′ le-Suc-ex
le-add2)

  have exec-4: execute ?M (0, tps0) (length es1 + length es2 + 2) = (0, tps2)
  proof −
    have execute ?M (0, tps0) (length es1 + length es2 + 2) = execute ?M (0, tps0) (Suc (length es1 + length
es2 + 1))
      by simp
    also have ... = exe ?M (execute ?M (0, tps0) (length es1 + length es2 + 1))
      by simp
    also have ... = exe ?M (execute M2 (0, tps1) (length es2) <+=> (length M1 + 1))
        (is - = exe ?M ?cfg)
      using exec-3′ by simp
    also have ... = sem (?M ! (fst ?cfg)) ?cfg
      using exe-def assms(2) len-M trace-def traces-def by auto
    also have ... = sem (cmd-jump (λ-. True) 0 0) ?cfg
    proof −
      have fst ?cfg = length M1 + length M2 + 1
        using assms(2) len-M trace-def traces-def by simp
      then have ?M ! (fst ?cfg) = cmd-jump (λ-. True) 0 0
        by (metis (no-types, lifting) add.right-neutral add-Suc-right length-Cons
          list.size(3) nth-append-length nth-append-length-plus parts plus-1-eq-Suc length-relocate)
      then show ?thesis
        by simp
    qed
    also have ... = (0, tps2)
      using assms(2) sem-jump trace-def traces-def by auto
    finally show ?thesis
      by simp
  qed

  show execute ?M (0, tps0) (length ?es) = (0, tps2)
    using exec-4 by auto
  show fst (execute ?M (0, tps0) i) < length ?M
      if i < length ?es for i
  proof −
    consider
       i < length es1
     | i = length es1
     | i ≥ length es1 + 1 and i ≤ length es1 + length es2 + 1
     | i = length es1 + length es2 + 2
     using ‹i < length ?es› by fastforce
    then show ?thesis
    proof (cases)
      case 1
      then have fst (execute ?M (0, tps0) i) = fst (execute M1 (0, tps0) i)
        using exec-1 by simp
      moreover have ∀ i<length es1. fst (execute M1 (0, tps0) i) < length M1
        using assms trace-def traces-def by simp
      ultimately have fst (execute ?M (0, tps0) i) < length M1
        using 1 by simp
      then show ?thesis
        using len-M by simp
    next
      case 2
      then have fst (execute ?M (0, tps0) i) = fst (execute M1 (0, tps0) i)
        using exec-1 by simp
      moreover have execute M1 (0, tps0) (length es1) = (length M1, tps1)
        using assms trace-def traces-def by simp
```

ultimately show *?thesis*
using *2* **by** (*simp add*: *len-M*)
next
case *3*
then have *eq*: *execute ?M* (*0*, *tps0*) *i* = *execute M2* (*0*, *tps1*) (*i* − (*length es1* + *1*)) <+=> (*length M1* + *1*)
using *exec-3* **by** *simp*
have *a*: ∀ *i*<*length es2*. *fst* (*execute M2* (*0*, *tps1*) *i*) < *length M2*
using *assms*(*2*) *trace-def traces-def that* **by** *simp*
have *b*: *fst* (*execute M2* (*0*, *tps1*) (*length es2*)) = *length M2*
using *assms*(*2*) *trace-def traces-def that* **by** *simp*
have *i* − (*length es1* + *1*) ≤ *length es2*
using *3* **by** *simp*
then have *fst* (*execute M2* (*0*, *tps1*) (*i* − (*length es1* + *1*))) ≤ *length M2*
using *a b that* **using** *le-eq-less-or-eq* **by** *auto*
then have *fst* (*execute M2* (*0*, *tps1*) (*i* − (*length es1* + *1*)) <+=> (*length M1* + *1*)) < *length ?M*
by (*simp add*: *len-M*)
then show *?thesis*
using *eq* **by** *simp*
next
case *4*
then show *?thesis*
using *exec-4* **using** *len-es that* **by** *linarith*
qed
qed
show *execute ?M* (*0*, *tps0*) (*Suc i*) <#> *0* = *fst* (*?es* ! *i*) ∧
execute ?M (*0*, *tps0*) (*Suc i*) <#> *1* = *snd* (*?es* ! *i*)
if *i* < *length ?es* **for** *i*
proof −
consider
i < *length es1*
| *i* = *length es1*
| *i* ≥ *length es1* + *1* **and** *i* < *length es1* + *length es2* + *1*
| *i* = *length es1* + *length es2* + *1*
using ‹*i* < *length ?es*› **by** *fastforce*
then show *?thesis*
proof (*cases*)
case *1*
then have *Suc i* ≤ *length es1*
by *simp*
then have *execute ?M* (*0*, *tps0*) (*Suc i*) = *execute M1* (*0*, *tps0*) (*Suc i*)
using *exec-1* **by** *blast*
then show *?thesis*
using *assms*(*1*) *trace-def traces-def* **by** (*simp add*: *1 nth-append*)
next
case *2*
then have *execute ?M* (*0*, *tps0*) (*Suc i*) = (*length M1* + *1*, *tps1*)
using *exec-2* **by** *simp*
then show *?thesis*
using *2* **by** *simp*
next
case *3*
then have *Suc-i*: *Suc i* ≥ *length es1* + *1* *Suc i* ≤ *length es1* + *length es2* + *1*
by *simp-all*
then have ∗: *execute ?M* (*0*, *tps0*) (*Suc i*) =
execute M2 (*0*, *tps1*) (*Suc i* − (*length es1* + *1*)) <+=> (*length M1* + *1*)
using *exec-3* **by** *blast*
from *3* **have** *i*: *i* − (*length es1* + *1*) < *length es2* (**is** *?j* < *length es2*)
by *simp*
then have ∗∗: *execute M2* (*0*, *tps1*) (*Suc ?j*) <#> *0* = *fst* (*es2* ! *?j*) ∧
execute M2 (*0*, *tps1*) (*Suc ?j*) <#> *1* = *snd* (*es2* ! *?j*)
using *assms*(*2*) *trace-def traces-def* **by** *simp*
have ((*es1* @ [(*tps1* :#: *0*, *tps1* :#: *1*)]) @ *es2*) ! *i* = *es2* ! *?j*

using *i 3* **by** (*simp add*: *nth-append*)
then have *es2* ! *?j* = *?es* ! *i*
by (*metis Suc-eq-plus1 append.assoc i length-append-singleton nth-append*)
then show *?thesis*
using ∗ ∗∗ **using** *3*(*1*) *Suc-diff-le* **by** *fastforce*
next
case *4*
then have *execute ?M* (*0*, *tps0*) (*Suc i*) = (*0*, *tps2*)
using *exec-4* **by** *simp*
then show *?thesis*
by (*simp add*: *4 nth-append*)
qed
qed
qed

lemma *tm-loop-sem-true-tracesI*:
assumes *traces M1 tps0 es1 tps1*
and *traces M2 tps1 es2 tps2*
and *cond* (*read tps1*)
and *es* = *es1* @ [(*tps1* :#: *0*, *tps1* :#: *1*)] @ *es2* @ [(*tps2* :#: *0*, *tps2* :#: *1*)]
shows *trace* (*WHILE M1* ; *cond DO M2 DONE*) (*0*, *tps0*) *es* (*0*, *tps2*)
using *assms tm-loop-sem-true-traces* **by** *blast*

Combining traces for m iterations of a loop. Typically m will be the total number of iterations.

lemma *tm-loop-trace-simple*:
fixes *m* :: *nat*
and *M* :: *machine*
and *tps* :: *nat* ⇒ *tape list*
and *es* :: *nat* ⇒ (*nat* × *nat*) *list*
assumes $\bigwedge$*i*. *i* < *m* ⟹ *trace M* (*0*, *tps i*) (*es i*) (*0*, *tps* (*Suc i*))
shows *trace M* (*0*, *tps 0*) (*concat* (*map es* [*0*..<*m*])) (*0*, *tps m*)
using *assms trace-Nil trace-additive* **by** (*induction m*) *simp-all*

For simple loops, where we have an upper bound for the length of traces independent of the iteration, there is a trivial upper bound for the length of the trace of m iterations. This is the only situation we will encounter.

lemma *length-concat-le*:
assumes $\bigwedge$*i*. *i* < *m* ⟹ *length* (*es i*) ≤ *b*
shows *length* (*concat* (*map es* [*0*..<*m*])) ≤ *m* ∗ *b*
using *assms*
proof (*induction m*)
case *0*
then show *?case*
by *simp*
next
case (*Suc m*)
have *length* (*concat* (*map es* [*0*..<*Suc m*])) = *length* (*concat* (*map es* [*0*..<*m*])) + *length* (*es m*)
by *simp*
also have ... ≤ *m* ∗ *b* + *length* (*es m*)
using *Suc* **by** *simp*
also have ... ≤ *m* ∗ *b* + *b*
using *Suc* **by** *simp*
also have ... = (*Suc m*) ∗ *b*
by *simp*
finally show *?case* **.**
qed

5.1.4 Traces for elementary Turing machines

Just like the not necessarily oblivious Turing machines considered so far, our oblivious Turing machines will be built from elementary ones from Section 2.4. In this subsection we show the traces of all the elementary machines we will need.

lemma *tm-left-0-traces*:
  assumes *length tps* > *1*
  shows *traces*
    (*tm-left 0*)
    tps
    [(*tps* :#: *0* − *1*, *tps* :#: *1*)]
    (*tps*[*0*:=(*fst* (*tps* ! *0*), *snd* (*tps* ! *0*) − *1*)])
proof −
  from *assms* **have** *length tps* > *0*
    by *auto*
  with *assms* **show** *?thesis*
    using *execute-tm-left assms tm-left-def* **by** (*intro tracesI*) *simp-all*
qed

lemma *traces-tm-left-0I*:
  assumes *length tps* > *1*
    and *es* = [(*tps* :#: *0* − *1*, *tps* :#: *1*)]
    and *tps′* = (*tps*[*0*:=(*fst* (*tps* ! *0*), *snd* (*tps* ! *0*) − *1*)])
  shows *traces* (*tm-left 0*) *tps es tps′*
  using *tm-left-0-traces assms* **by** *simp*

lemma *tm-left-1-traces*:
  assumes *length tps* > *1*
  shows *traces*
    (*tm-left 1*)
    tps
    [(*tps* :#: *0*, *tps* :#: *1* − *1*)]
    (*tps*[*1*:=(*fst* (*tps* ! *1*), *snd* (*tps* ! *1*) − *1*)])
proof −
  from *assms* **have** *length tps* > *0*
    by *auto*
  with *assms* **show** *?thesis*
    using *execute-tm-left assms tm-left-def* **by** (*intro tracesI*) *simp-all*
qed

lemma *traces-tm-left-1I*:
  assumes *length tps* > *1*
    and *es* = [(*tps* :#: *0*, *tps* :#: *1* − *1*)]
    and *tps′* = (*tps*[*1*:=(*fst* (*tps* ! *1*), *snd* (*tps* ! *1*) − *1*)])
  shows *traces* (*tm-left 1*) *tps es tps′*
  using *tm-left-1-traces assms* **by** *simp*

lemma *tm-right-0-traces*:
  assumes *length tps* > *1*
  shows *traces*
    (*tm-right 0*)
    tps
    [(*tps* :#: *0* + *1*, *tps* :#: *1*)]
    (*tps*[*0*:=(*fst* (*tps* ! *0*), *snd* (*tps* ! *0*) + *1*)])
proof −
  from *assms* **have** *length tps* > *0*
    by *auto*
  with *assms* **show** *?thesis*
    using *execute-tm-right assms tm-right-def* **by** (*intro tracesI*) *simp-all*
qed

lemma *traces-tm-right-0I*:
  assumes *length tps* > *1*
    and *es* = [(*tps* :#: *0* + *1*, *tps* :#: *1*)]
    and *tps′* = (*tps*[*0*:=(*fst* (*tps* ! *0*), *snd* (*tps* ! *0*) + *1*)])
  shows *traces* (*tm-right 0*) *tps es tps′*
  using *tm-right-0-traces assms* **by** *simp*

lemma *tm-right-1-traces*:
  assumes *length tps* > *1*
  shows *traces*
    (*tm-right 1*)
    tps
    [(*tps* :#: *0*, *tps* :#: *1* + *1*)]
    (*tps*[*1*:=(*fst* (*tps* ! *1*), *snd* (*tps* ! *1*) + *1*)])
proof −
  from *assms* **have** *length tps* > *0*
    by *auto*
  with *assms* **show** *?thesis*
    using *execute-tm-right assms tm-right-def* **by** (*intro tracesI*) *simp-all*
qed

lemma *tm-rtrans-1-traces*:
  assumes *1* < *length tps*
  shows *traces*
    (*tm-rtrans 1 f*)
    tps
    [(*tps* :#: *0*, *tps* :#: *1* + *1*)]
    (*tps*[*1* := *tps* ! *1* |:=| *f* (*tps* :.: *1*) |+| *1*])
  using *execute-tm-rtrans assms tm-rtrans-def* **by** (*intro tracesI*) *simp-all*

lemma *traces-tm-right-1I*:
  assumes *length tps* > *1*
    and *es* = [(*tps* :#: *0*, *tps* :#: *1* + *1*)]
    and *tps′* = (*tps*[*1*:=(*fst* (*tps* ! *1*), *snd* (*tps* ! *1*) + *1*)])
  shows *traces* (*tm-right 1*) *tps es tps′*
  using *tm-right-1-traces assms* **by** *simp*

lemma *traces-tm-rtrans-1I*:
  assumes *1* < *length tps*
    and *es* = [(*tps* :#: *0*, *tps* :#: *1* + *1*)]
    and *tps′* = (*tps*[*1* := *tps* ! *1* |:=| *f* (*tps* :.: *1*) |+| *1*])
  shows *traces* (*tm-rtrans 1 f*) *tps es tps′*
  using *tm-rtrans-1-traces assms* **by** *simp*

lemma *tm-left-until-1-traces*:
  assumes *length tps* > *1* **and** *begin-tape H* (*tps* ! *1*)
  shows *traces*
    (*tm-left-until H 1*)
    tps
    (*map* (λ*i*. (*tps* :#: *0*, *i*)) (*rev* [*0*..<*tps* :#: *1*]) @ [(*tps* :#: *0*, *0*)])
    (*tps*[*1* := *tps* ! *1* |#=| *0*])
proof
  let *?es* = *map* (λ*i*. (*tps* :#: *0*, *i*)) (*rev* [*0*..<*tps* :#: *1*]) @ [(*tps* :#: *0*, *0*)]
  show *execute* (*tm-left-until H 1*) (*0*, *tps*) (*length ?es*) = (*length* (*tm-left-until H 1*), *tps*[*1* := *tps* ! *1* |#=| *0*])
    using *execute-tm-left-until assms tm-left-until-def* **by** *simp*
  show ⋀*i*. *i* < *length ?es* ⟹ *fst* (*execute* (*tm-left-until H 1*) (*0*, *tps*) *i*) < *length* (*tm-left-until H 1*)
    using *execute-tm-left-until-less assms tm-left-until-def* **by** *simp*
  show *execute* (*tm-left-until H 1*) (*0*, *tps*) (*Suc i*) <#> *0* = *fst* (*?es* ! *i*) ∧
        execute (*tm-left-until H 1*) (*0*, *tps*) (*Suc i*) <#> *1* = *snd* (*?es* ! *i*)
      if *i* < *length ?es* **for** *i*
  proof (*cases i* < *tps* :#: *1*)
    case *True*
    then have *i*: *Suc i* ≤ *tps* :#: *1*
      by *simp*
    then have *execute* (*tm-left-until H 1*) (*0*, *tps*) (*Suc i*) = (*0*, *tps*[*1* := *tps* ! *1* |−| *Suc i*])
      using *execute-tm-left-until-less assms* **by** *presburger*
    moreover have *?es* ! *i* = (*tps* :#: *0*, *tps* :#: *1* − *Suc i*)
    proof −
      have *?es* ! *i* = (*map* (λ*i*. (*tps* :#: *0*, *i*)) (*rev* [*0*..<*tps* :#: *1*])) ! *i*
        using *True* **by** (*simp add*: *nth-append*)

moreover have (*rev* [*0*..<*tps* :#: *1*]) ! *i* = *tps* :#: *1* − *Suc i*
  using *True* **by** (*simp add*: *rev-nth*)
ultimately show *?thesis*
  using *True* **by** *simp*
qed
ultimately show *?thesis*
  using *assms*(*1*) **by** *simp*
next
  case *False*
  then have *i*: *i* = *tps* :#: *1*
    using *that* **by** *simp*
  then have *execute* (*tm-left-until H 1*) (*0*, *tps*) (*Suc* (*tps* :#: *1*)) = (*1*, *tps*[*1* := *tps* ! *1* |#=| *0*])
    using *execute-tm-left-until assms* **by** *simp*
  then have *execute* (*tm-left-until H 1*) (*0*, *tps*) (*Suc i*) = (*1*, *tps*[*1* := *tps* ! *1* |#=| *0*])
    using *i* **by** *simp*
  moreover have *?es* ! *i* = (*tps* :#: *0*, *0*)
    using *i* **by** (*metis diff-zero length-map length-rev length-upt nth-append-length*)
  ultimately show *?thesis*
    using *assms*(*1*) **by** *simp*
qed
qed

lemma *traces-tm-left-until-1I*:
  assumes *length tps* > *1*
    and *begin-tape H* (*tps* ! *1*)
    and *es* = *map* (*λi.* (*tps* :#: *0*, *i*)) (*rev* [*0*..<*tps* :#: *1*]) @ [(*tps* :#: *0*, *0*)]
    and *tps'* = *tps*[*1* := *tps* ! *1* |#=| *0*]
  shows *traces* (*tm-left-until H 1*) *tps es tps'*
  using *tm-left-until-1-traces assms* **by** *simp*

lemma *tm-left-until-0-traces*:
  assumes *length tps* > *1* **and** *begin-tape H* (*tps* ! *0*)
  shows *traces*
    (*tm-left-until H 0*)
    tps
    (*map* (*λi.* (*i*, *tps* :#: *1*)) (*rev* [*0*..<*tps* :#: *0*]) @ [(*0*, *tps* :#: *1*)])
    (*tps*[*0* := *tps* ! *0* |#=| *0*])
proof
  have *len*: *length tps* > *0*
    using *assms*(*1*) **by** *auto*
  let *?es* = *map* (*λi.* (*i*, *tps* :#: *1*)) (*rev* [*0*..<*tps* :#: *0*]) @ [(*0*, *tps* :#: *1*)]
  show *execute* (*tm-left-until H 0*) (*0*, *tps*) (*length ?es*) = (*length* (*tm-left-until H 0*), *tps*[*0* := *tps* ! *0* |#=| *0*])
    using *execute-tm-left-until assms*(*2*) *tm-left-until-def len* **by** *simp*
  show $\bigwedge$*i. i* < *length ?es* ⟹ *fst* (*execute* (*tm-left-until H 0*) (*0*, *tps*) *i*) < *length* (*tm-left-until H 0*)
    using *execute-tm-left-until-less len assms*(*2*) *tm-left-until-def* **by** *simp*
  show *execute* (*tm-left-until H 0*) (*0*, *tps*) (*Suc i*) <#> *0* = *fst* (*?es* ! *i*) ∧
      execute (*tm-left-until H 0*) (*0*, *tps*) (*Suc i*) <#> *1* = *snd* (*?es* ! *i*)
    if *i* < *length ?es* **for** *i*
  proof (*cases i* < *tps* :#: *0*)
    case *True*
    then have *i*: *Suc i* ≤ *tps* :#: *0*
      by *simp*
    then have *execute* (*tm-left-until H 0*) (*0*, *tps*) (*Suc i*) = (*0*, *tps*[*0* := *tps* ! *0* |−| *Suc i*])
      using *execute-tm-left-until-less assms*(*2*) *len* **by** *blast*
    moreover have *?es* ! *i* = (*tps* :#: *0* − *Suc i*, *tps* :#: *1*)
    proof −
      have *?es* ! *i* = (*map* (*λi.* (*i*, *tps* :#: *1*)) (*rev* [*0*..<*tps* :#: *0*])) ! *i*
        using *True* **by** (*simp add*: *nth-append*)
      moreover have (*rev* [*0*..<*tps* :#: *0*]) ! *i* = *tps* :#: *0* − *Suc i*
        using *True* **by** (*simp add*: *rev-nth*)
      ultimately show *?thesis*
        using *True* **by** *simp*
    qed

```
    ultimately show ?thesis
      using len by simp
  next
    case False
    then have i: i = tps :#: 0
      using that by simp
    then have execute (tm-left-until H 0) (0, tps) (Suc (tps :#: 0)) = (1, tps[0 := tps ! 0 |#=| 0])
      using execute-tm-left-until assms len by simp
    then have execute (tm-left-until H 0) (0, tps) (Suc i) = (1, tps[0 := tps ! 0 |#=| 0])
      using i by simp
    moreover have ?es ! i = (0, tps :#: 1)
       using i by (metis One-nat-def diff-Suc-1 diff-Suc-Suc length-map length-rev length-upt nth-append-length)
    ultimately show ?thesis
      using len by simp
  qed
qed

lemma traces-tm-left-until-0I:
  assumes length tps > 1
    and begin-tape H (tps ! 0)
    and es = map (λi. (i, tps :#: 1)) (rev [0..<tps :#: 0]) @ [(0, tps :#: 1)]
    and tps′ = tps[0 := tps ! 0 |#=| 0]
  shows traces (tm-left-until H 0) tps es tps′
  using tm-left-until-0-traces assms by simp

lemma tm-start-0-traces:
  assumes length tps > 1 and clean-tape (tps ! 0)
  shows traces
    (tm-start 0)
    tps
    (map (λi. (i, tps :#: 1)) (rev [0..<tps :#: 0]) @ [(0, tps :#: 1)])
    (tps[0 := tps ! 0 |#=| 0])
proof −
  have begin-tape {1} (tps ! 0)
    using clean-tape-def begin-tape-def assms(2) by simp
  then show ?thesis
    using tm-left-until-0-traces tm-start-def assms(1) by metis
qed

lemma tm-start-1-traces:
  assumes length tps > 1 and clean-tape (tps ! 1)
  shows traces
    (tm-start 1)
    tps
    (map (λi. (tps :#: 0, i)) (rev [0..<tps :#: 1]) @ [(tps :#: 0, 0)])
    (tps[1 := tps ! 1 |#=| 0])
proof −
  have begin-tape {1} (tps ! 1)
    using clean-tape-def begin-tape-def assms(2) by simp
  then show ?thesis
    using tm-left-until-1-traces tm-start-def assms(1) by metis
qed

lemma traces-tm-start-1I:
  assumes length tps > 1
    and clean-tape (tps ! 1)
    and es = map (λi. (tps :#: 0, i)) (rev [0..<tps :#: 1]) @ [(tps :#: 0, 0)]
    and tps′ = tps[1 := tps ! 1 |#=| 0]
  shows traces (tm-start 1) tps es tps′
  using tm-start-1-traces assms by simp

lemma tm-cr-0-traces:
  assumes length tps > 1 and clean-tape (tps ! 0)
```

```
  shows traces
    (tm-cr 0)
    tps
    ((map (λi. (i, tps :#: 1)) (rev [0..<tps :#: 0]) @ [(0, tps :#: 1)]) @ [(1, tps :#: 1)])
    (tps[0 := tps ! 0 |#=| 1])
  unfolding tm-cr-def
proof (rule traces-sequential[where ?tps2.0=tps[0 := tps ! 0 |#=| 0]])
  from assms(1) have len: length tps > 0
    by auto
  show traces (tm-start 0) tps
     (map (λi. (i, tps :#: 1)) (rev [0..<tps :#: 0]) @ [(0, tps :#: 1)])
     (tps[0 := tps ! 0 |#=| 0])
    using assms tm-start-0-traces by simp
  show traces (tm-right 0) (tps[0 := tps ! 0 |#=| 0])
     [(1, tps :#: 1)] (tps[0 := tps ! 0 |#=| 1])
    using assms tm-right-0-traces len
    by (smt (verit) One-nat-def add.commute fst-conv length-list-update list-update-overwrite neq0-conv
      nth-list-update-eq nth-list-update-neq plus-1-eq-Suc snd-conv zero-less-one)
qed

lemma traces-tm-cr-0I:
  assumes length tps > 1 and clean-tape (tps ! 0)
    and es = map (λi. (i, tps :#: 1)) (rev [0..<tps :#: 0]) @ [(0, tps :#: 1), (1, tps :#: 1)]
    and tps' = tps[0 := tps ! 0 |#=| 1]
  shows traces (tm-cr 0) tps es tps'
  using tm-cr-0-traces assms by simp

lemma tm-cr-1-traces:
  assumes length tps > 1 and clean-tape (tps ! 1)
  shows traces
    (tm-cr 1)
    tps
    ((map (λi. (tps :#: 0, i)) (rev [0..<tps :#: 1]) @ [(tps :#: 0, 0)]) @ [(tps :#: 0, 1)])
    (tps[1 := tps ! 1 |#=| 1])
  unfolding tm-cr-def
proof (rule traces-sequential[where ?tps2.0=tps[1 := tps ! 1 |#=| 0]])
  show traces (tm-start 1) tps
     (map (Pair (tps :#: 0)) (rev [0..<tps :#: 1]) @ [(tps :#: 0, 0)])
     (tps[1 := tps ! 1 |#=| 0])
    using assms tm-start-1-traces by simp
  show traces (tm-right 1) (tps[1 := tps ! 1 |#=| 0])
     [(tps :#: 0, 1)] (tps[1 := tps ! 1 |#=| 1])
    using assms tm-right-1-traces
    by (smt (verit) One-nat-def add.commute fst-conv length-list-update list-update-overwrite neq0-conv
      nth-list-update-eq nth-list-update-neq plus-1-eq-Suc snd-conv zero-less-one)
qed

lemma traces-tm-cr-1I:
  assumes length tps > 1 and clean-tape (tps ! 1)
    and es = map (λi. (tps :#: 0, i)) (rev [0..<tps :#: 1]) @ [(tps :#: 0, 0), (tps :#: 0, 1)]
    and tps' = tps[1 := tps ! 1 |#=| 1]
  shows traces (tm-cr 1) tps es tps'
  using tm-cr-1-traces assms by simp

lemma heads-tm-trans-until-less:
  assumes j1 < k j2 < k and length tps = k
    and rneigh (tps ! j1) H n
    and t ≤ n
  shows execute (tm-trans-until j1 j2 H f) (0, tps) t <#> j1 = tps :#: j1 + t
    and execute (tm-trans-until j1 j2 H f) (0, tps) t <#> j2 = tps :#: j2 + t
  using assms execute-tm-trans-until-less[OF assms]
  by ((metis (no-types, lifting) length-list-update nth-list-update-eq nth-list-update-neq sndI transplant-def),
      (metis (no-types, lifting) length-list-update nth-list-update-eq sndI transplant-def))
```

lemma *heads-tm-ltrans-until-less*:
  assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* = *k*
    and *lneigh* (*tps* ! *j1*) *H n*
    and $t \leq n$
    and $n \leq$ *tps* :#: *j1*
    and $n \leq$ *tps* :#: *j2*
  shows *execute* (*tm-ltrans-until j1 j2 H f*) (*0*, *tps*) *t* <#> *j1* = *tps* :#: *j1* − *t*
    and *execute* (*tm-ltrans-until j1 j2 H f*) (*0*, *tps*) *t* <#> *j2* = *tps* :#: *j2* − *t*
  using *assms execute-tm-ltrans-until-less*[*OF assms*]
  by ((*metis* (*no-types*, *lifting*) *length-list-update nth-list-update-eq nth-list-update-neq sndI ltransplant-def*),
      (*metis* (*no-types*, *lifting*) *length-list-update nth-list-update-eq sndI ltransplant-def*))

lemma *heads-tm-trans-until-less′*:
  assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* = *k*
    and *rneigh* (*tps* ! *j1*) *H n*
    and $t \leq n$
    and $j \neq j1$ **and** $j \neq j2$
  shows *execute* (*tm-trans-until j1 j2 H f*) (*0*, *tps*) *t* <#> *j* = *tps* :#: *j*
  using *assms execute-tm-trans-until-less*[*OF assms*(*1*−*5*)] **by** *simp*

lemma *heads-tm-ltrans-until-less′*:
  assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* = *k*
    and *lneigh* (*tps* ! *j1*) *H n*
    and $t \leq n$
    and $n \leq$ *tps* :#: *j1*
    and $n \leq$ *tps* :#: *j2*
    and $j \neq j1$ **and** $j \neq j2$
  shows *execute* (*tm-ltrans-until j1 j2 H f*) (*0*, *tps*) *t* <#> *j* = *tps* :#: *j*
  using *assms execute-tm-ltrans-until-less*[*OF assms*(*1*−*7*)] **by** *simp*

lemma *heads-tm-trans-until*:
  assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* = *k* **and** *rneigh* (*tps* ! *j1*) *H n*
  shows *execute* (*tm-trans-until j1 j2 H f*) (*0*, *tps*) (*Suc n*) <#> *j1* = *tps* :#: *j1* + *n*
    and *execute* (*tm-trans-until j1 j2 H f*) (*0*, *tps*) (*Suc n*) <#> *j2* = *tps* :#: *j2* + *n*
  using *assms execute-tm-trans-until*[*OF assms*]
    by ((*metis* (*no-types*, *lifting*) *length-list-update nth-list-update-eq nth-list-update-neq snd-conv transplant-def*),
      (*metis* (*no-types*, *lifting*) *length-list-update nth-list-update-eq snd-conv transplant-def*))

lemma *heads-tm-ltrans-until*:
  assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* = *k*
    and *lneigh* (*tps* ! *j1*) *H n*
    and $n \leq$ *tps* :#: *j1*
    and $n \leq$ *tps* :#: *j2*
  shows *execute* (*tm-ltrans-until j1 j2 H f*) (*0*, *tps*) (*Suc n*) <#> *j1* = *tps* :#: *j1* − *n*
    and *execute* (*tm-ltrans-until j1 j2 H f*) (*0*, *tps*) (*Suc n*) <#> *j2* = *tps* :#: *j2* − *n*
  using *assms execute-tm-ltrans-until*[*OF assms*]
    by ((*metis* (*no-types*, *lifting*) *length-list-update nth-list-update-eq nth-list-update-neq snd-conv ltransplant-def*),
      (*metis* (*no-types*, *lifting*) *length-list-update nth-list-update-eq snd-conv ltransplant-def*))

lemma *heads-tm-trans-until′*:
  assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* = *k*
    and *rneigh* (*tps* ! *j1*) *H n*
    and $j \neq j1$ **and** $j \neq j2$
  shows *execute* (*tm-trans-until j1 j2 H f*) (*0*, *tps*) (*Suc n*) <#> *j* = *tps* :#: *j*
  using *assms execute-tm-trans-until*[*OF assms*(*1*−*4*)] **by** *simp*

lemma *heads-tm-ltrans-until′*:
  assumes $j1 < k$ **and** $j2 < k$ **and** *length tps* = *k*
    and *lneigh* (*tps* ! *j1*) *H n*
    and $n \leq$ *tps* :#: *j1*
    and $n \leq$ *tps* :#: *j2*
    and $j \neq j1$ **and** $j \neq j2$

shows *execute* (*tm-ltrans-until j1 j2 H f*) (*0*, *tps*) (*Suc n*) <#> *j* = *tps* :#: *j*
using *assms execute-tm-ltrans-until*[*OF assms*(*1*−*6*)] **by** *simp*

lemma *traces-tm-trans-until-11*:
assumes *1* < *k* **and** *length tps* = *k* **and** *rneigh* (*tps* ! *1*) *H n*
shows *traces* (*tm-trans-until 1 1 H f*)
tps
(*map* (λ*i*. (*tps* :#: *0*, *tps* :#: *1* + *Suc i*)) [*0*..<*n*] @ [(*tps* :#: *0*, *tps* :#: *1* + *n*)])
(*tps*[*1* := *transplant* (*tps* ! *1*) (*tps* ! *1*) *f n*])
proof
let *?es* = *map* (λ*i*. (*tps* :#: *0*, *tps* :#: *1* + *Suc i*)) [*0*..<*n*] @ [(*tps* :#: *0*, *tps* :#: *1* + *n*)]
let *?tps* = *tps* [*1* := *transplant* (*tps* ! *1*) (*tps* ! *1*) *f n*]
let *?M* = *tm-trans-until 1 1 H f*
have *len*: *length ?es* = *Suc n*
by *simp*
show *execute ?M* (*0*, *tps*) (*length ?es*) = (*length ?M*, *?tps*)
using *tm-trans-until-def len execute-tm-trans-until assms* **by** *simp*
show *fst* (*execute ?M* (*0*, *tps*) *i*) < *length ?M* **if** *i* < *length ?es* **for** *i*
proof −
from *that len* **have** *i* ≤ *n*
by *simp*
then have *fst* (*execute ?M* (*0*, *tps*) *i*) = *0*
using *execute-tm-trans-until-less assms* **by** *simp*
then show *?thesis*
using *tm-trans-until-def* **by** *simp*
qed
show *execute ?M* (*0*, *tps*) (*Suc i*) <#> *0* = *fst* (*?es* ! *i*) ∧
execute ?M (*0*, *tps*) (*Suc i*) <#> *1* = *snd* (*?es* ! *i*)
if *i* < *length ?es* **for** *i*
proof (*cases i* < *n*)
case *True*
then have *?es* ! *i* = (*tps* :#: *0*, *tps* :#: *1* + *Suc i*)
by (*simp add*: *nth-append*)
moreover from *True* **have** *Suc i* ≤ *n*
by *simp*
ultimately show *?thesis*
using *heads-tm-trans-until-less′ heads-tm-trans-until-less assms*
by (*metis One-nat-def Suc-neq-Zero fst-conv snd-conv*)
next
case *False*
then have *i* = *n*
using *that* **by** *simp*
then have *?es* ! *i* = (*tps* :#: *0*, *tps* :#: *1* + *n*)
by (*metis* (*no-types*, *lifting*) *diff-zero length-map length-upt nth-append-length*)
then show *?thesis*
using *heads-tm-trans-until′ heads-tm-trans-until assms* ‹*i* = *n*› **by** *simp*
qed
qed

lemma *traces-tm-ltrans-until-11*:
assumes *1* < *k* **and** *length tps* = *k* **and** *lneigh* (*tps* ! *1*) *H n* **and** *n* ≤ *tps* :#: *1*
shows *traces* (*tm-ltrans-until 1 1 H f*)
tps
(*map* (λ*i*. (*tps* :#: *0*, *tps* :#: *1* − *Suc i*)) [*0*..<*n*] @ [(*tps* :#: *0*, *tps* :#: *1* − *n*)])
(*tps*[*1* := *ltransplant* (*tps* ! *1*) (*tps* ! *1*) *f n*])
proof
let *?es* = *map* (λ*i*. (*tps* :#: *0*, *tps* :#: *1* − *Suc i*)) [*0*..<*n*] @ [(*tps* :#: *0*, *tps* :#: *1* − *n*)]
let *?tps* = *tps* [*1* := *ltransplant* (*tps* ! *1*) (*tps* ! *1*) *f n*]
let *?M* = *tm-ltrans-until 1 1 H f*
have *len*: *length ?es* = *Suc n*
by *simp*
show *execute ?M* (*0*, *tps*) (*length ?es*) = (*length ?M*, *?tps*)
using *tm-ltrans-until-def len execute-tm-ltrans-until assms* **by** *simp*

```
  show fst (execute ?M (0, tps) i) < length ?M if i < length ?es for i
  proof -
    from that len have i ≤ n
      by simp
    then have fst (execute ?M (0, tps) i) = 0
      using execute-tm-ltrans-until-less assms by simp
    then show ?thesis
      using tm-ltrans-until-def by simp
  qed
  show execute ?M (0, tps) (Suc i) <#> 0 = fst (?es ! i) ∧
      execute ?M (0, tps) (Suc i) <#> 1 = snd (?es ! i)
    if i < length ?es for i
  proof (cases i < n)
    case True
    then have ?es ! i = (tps :#: 0, tps :#: 1 − Suc i)
      by (simp add: nth-append)
    moreover from True have Suc i ≤ n
      by simp
    ultimately show ?thesis
      using heads-tm-ltrans-until-less' heads-tm-ltrans-until-less assms
      by (metis One-nat-def Suc-neq-Zero fst-conv snd-conv)
  next
    case False
    then have i = n
      using that by simp
    then have ?es ! i = (tps :#: 0, tps :#: 1 − n)
      by (metis (no-types, lifting) diff-zero length-map length-upt nth-append-length)
    then show ?thesis
      using heads-tm-ltrans-until' heads-tm-ltrans-until assms ‹i = n› by simp
  qed
qed

lemma traces-tm-trans-until-01:
  assumes 0 < k and 1 < k and length tps = k and rneigh (tps ! 0) H n
  shows traces (tm-trans-until 0 1 H f)
    tps
    (map (λi. (tps :#: 0 + Suc i, tps :#: 1 + Suc i)) [0..<n] @ [(tps :#: 0 + n, tps :#: 1 + n)])
    (tps[0 := tps ! 0 |+| n, 1 := transplant (tps ! 0) (tps ! 1) f n])
proof
  let ?es = map (λi. (tps :#: 0 + Suc i, tps :#: 1 + Suc i)) [0..<n] @ [(tps :#: 0 + n, tps :#: 1 + n)]
  let ?tps = tps [0 := tps ! 0 |+| n, 1 := transplant (tps ! 0) (tps ! 1) f n]
  let ?M = tm-trans-until 0 1 H f
  have len: length ?es = Suc n
    by simp
  show execute ?M (0, tps) (length ?es) = (length ?M, ?tps)
    using tm-trans-until-def len execute-tm-trans-until[of 0 k 1] assms by simp
  show fst (execute ?M (0, tps) i) < length ?M if i < length ?es for i
  proof -
    from that len have i ≤ n
      by simp
    then have fst (execute ?M (0, tps) i) = 0
      using execute-tm-trans-until-less[of 0 k 1] assms by simp
    then show ?thesis
      using tm-trans-until-def by simp
  qed
  show execute ?M (0, tps) (Suc i) <#> 0 = fst (?es ! i) ∧
      execute ?M (0, tps) (Suc i) <#> 1 = snd (?es ! i)
    if i < length ?es for i
  proof (cases i < n)
    case True
    then have ?es ! i = (tps :#: 0 + Suc i, tps :#: 1 + Suc i)
      by (simp add: nth-append)
    moreover from True have Suc i ≤ n
```

by *simp*
ultimately show *?thesis*
using *heads-tm-trans-until-less*[*of 0 k 1 tps H n Suc i f*] *assms* **by** *simp*
next
case *False*
then have $i = n$
using *that* **by** *simp*
then have *?es* ! i = (*tps* :#: *0* + *n*, *tps* :#: *1* + *n*)
by (*metis* (*no-types*, *lifting*) *diff-zero length-map length-upt nth-append-length*)
then show *?thesis*
using *heads-tm-trans-until*[*of 0 k 1 tps H n f*] *assms* ‹$i = n$› **by** *simp*
qed
qed

lemma *traces-tm-trans-until-01I*:
assumes *1* < *length tps*
and *rneigh* (*tps* ! *0*) *H n*
and *es* = *map* (λ*i*. (*tps* :#: *0* + *Suc i*, *tps* :#: *1* + *Suc i*)) [*0*..<*n*] @ [(*tps* :#: *0* + *n*, *tps* :#: *1* + *n*)]
and *tps'* = *tps*[*0* := *tps* ! *0* |+| *n*, *1* := *transplant* (*tps* ! *0*) (*tps* ! *1*) *f n*]
shows *traces* (*tm-trans-until 0 1 H f*) *tps es tps'*
using *assms traces-tm-trans-until-01* **by** *simp*

lemma *traces-tm-trans-until-11I*:
assumes *1* < *length tps*
and *rneigh* (*tps* ! *1*) *H n*
and *es* = *map* (λ*i*. (*tps* :#: *0*, *tps* :#: *1* + *Suc i*)) [*0*..<*n*] @ [(*tps* :#: *0*, *tps* :#: *1* + *n*)]
and *tps'* = *tps*[*1* := *transplant* (*tps* ! *1*) (*tps* ! *1*) *f n*]
shows *traces* (*tm-trans-until 1 1 H f*) *tps es tps'*
using *assms traces-tm-trans-until-11* **by** *simp*

lemma *traces-tm-ltrans-until-11I*:
assumes *1* < *length tps* **and** ∀ *h*<*G*. *f h* < *G*
and *lneigh* (*tps* ! *1*) *H n*
and *n* ≤ *tps* :#: *1*
and *es* = *map* (λ*i*. (*tps* :#: *0*, *tps* :#: *1* − *Suc i*)) [*0*..<*n*] @ [(*tps* :#: *0*, *tps* :#: *1* − *n*)]
and *tps'* = *tps*[*1* := *ltransplant* (*tps* ! *1*) (*tps* ! *1*) *f n*]
shows *traces* (*tm-ltrans-until 1 1 H f*) *tps es tps'*
using *assms traces-tm-ltrans-until-11* **by** *simp*

lemma *traces-tm-const-until-01I*:
assumes *1* < *length tps*
and *rneigh* (*tps* ! *0*) *H n*
and *es* = *map* (λ*i*. (*tps* :#: *0* + *Suc i*, *tps* :#: *1* + *Suc i*)) [*0*..<*n*] @ [(*tps* :#: *0* + *n*, *tps* :#: *1* + *n*)]
and *tps'* = *tps*[*0* := *tps* ! *0* |+| *n*, *1* := *constplant* (*tps* ! *1*) *h n*]
shows *traces* (*tm-const-until 0 1 H h*) *tps es tps'*
using *assms traces-tm-trans-until-01 tm-const-until-def constplant-transplant*[*of - - - tps* ! *0*] **by** *simp*

lemma *traces-tm-const-until-11I*:
assumes *1* < *length tps* **and** *h* < *G*
and *rneigh* (*tps* ! *1*) *H n*
and *es* = *map* (λ*i*. (*tps* :#: *0*, *tps* :#: *1* + *Suc i*)) [*0*..<*n*] @ [(*tps* :#: *0*, *tps* :#: *1* + *n*)]
and *tps'* = *tps*[*1* := *constplant* (*tps* ! *1*) *h n*]
shows *traces* (*tm-const-until 1 1 H h*) *tps es tps'*
using *assms traces-tm-trans-until-11 tm-const-until-def constplant-transplant*[*of - - - tps* ! *1*] **by** *simp*

lemma *traces-tm-cp-until-01I*:
assumes *1* < *length tps*
and *rneigh* (*tps* ! *0*) *H n*
and *es* = *map* (λ*i*. (*tps* :#: *0* + *Suc i*, *tps* :#: *1* + *Suc i*)) [*0*..<*n*] @ [(*tps* :#: *0* + *n*, *tps* :#: *1* + *n*)]
and *tps'* = *tps*[*0* := *tps* ! *0* |+| *n*, *1* := *implant* (*tps* ! *0*) (*tps* ! *1*) *n*]
shows *traces* (*tm-cp-until 0 1 H*) *tps es tps'*
using *assms traces-tm-trans-until-01 tm-cp-until-def* **by** *simp*

lemma *traces-tm-cp-until-11I*:
  assumes $1 < length\ tps$
    and $rneigh\ (tps\ !\ 1)\ H\ n$
    and $es = map\ (\lambda i.\ (tps\ :\#:\ 0,\ tps\ :\#:\ 1 + Suc\ i))\ [0..<n]\ @\ [(tps\ :\#:\ 0,\ tps\ :\#:\ 1 + n)]$
    and $tps' = tps[1 := implant\ (tps\ !\ 1)\ (tps\ !\ 1)\ n]$
  shows $traces\ (tm\text{-}cp\text{-}until\ 1\ 1\ H)\ tps\ es\ tps'$
  using *assms traces-tm-trans-until-11 tm-cp-until-def* **by** *simp*

lemma *traces-tm-right-until-1I*:
  assumes $1 < length\ tps$
    and $rneigh\ (tps\ !\ 1)\ H\ n$
    and $es = map\ (\lambda i.\ (tps\ :\#:\ 0,\ tps\ :\#:\ 1 + Suc\ i))\ [0..<n]\ @\ [(tps\ :\#:\ 0,\ tps\ :\#:\ 1 + n)]$
    and $tps' = tps[1 := (tps\ !\ 1)\ |+|\ n]$
  shows $traces\ (tm\text{-}right\text{-}until\ 1\ H)\ tps\ es\ tps'$
  using *assms traces-tm-cp-until-11I tm-right-until-def implant-self* **by** *simp*

lemma *execute-tm-write*:
  shows $execute\ (tm\text{-}write\ j\ h)\ (0,\ tps)\ 1 = (1,\ tps[j := tps\ !\ j\ |{:=}|\ h])$
  using *sem-cmd-write exe-lt-length tm-write-def* **by** *simp*

lemma *traces-tm-writeI*:
  assumes $j > 0$ **and** $j < length\ tps$
    and $es = [(tps\ :\#:\ 0,\ tps\ :\#:\ 1)]$
    and $tps' = tps[j := tps\ !\ j\ |{:=}|\ h]$
  shows $traces\ (tm\text{-}write\ j\ h)\ tps\ es\ tps'$
  using *assms execute-tm-write tm-write-def* **by** (*intro tracesI*) (*auto simp add*: *nth-list-update*)

corollary *traces-tm-write-1I*:
  assumes $1 < length\ tps$
    and $es = [(tps\ :\#:\ 0,\ tps\ :\#:\ 1)]$
    and $tps' = tps[1 := tps\ !\ 1\ |{:=}|\ h]$
  shows $traces\ (tm\text{-}write\ 1\ h)\ tps\ es\ tps'$
  using *assms traces-tm-writeI* **by** *simp*

corollary *traces-tm-write-ge2I*:
  assumes $j \geq 2$
    and $j < length\ tps$
    and $es = [(tps\ :\#:\ 0,\ tps\ :\#:\ 1)]$
    and $tps' = tps[j := tps\ !\ j\ |{:=}|\ h]$
  shows $traces\ (tm\text{-}write\ j\ h)\ tps\ es\ tps'$
  using *assms traces-tm-writeI* **by** *simp*

lemma *execute-tm-write-manyI*:
  assumes $0 \notin J$ **and** $\forall j{\in}J.\ j < k$ **and** $k \geq 2$ **and** $length\ tps = k$
    and $length\ tps' = k$
    and $\bigwedge j.\ j \in J \Longrightarrow tps'\ !\ j = tps\ !\ j\ |{:=}|\ h$
    and $\bigwedge j.\ j < k \Longrightarrow j \notin J \Longrightarrow tps'\ !\ j = tps\ !\ j$
  shows $execute\ (tm\text{-}write\text{-}many\ J\ h)\ (0,\ tps)\ 1 = (1,\ tps')$
proof −
  have $tps' = map\ (\lambda j.\ if\ j \in J\ then\ tps\ !\ j\ |{:=}|\ h\ else\ tps\ !\ j)\ [0..<length\ tps]$ (**is** - = *?rhs*)
    using *assms* **by** (*intro nth-equalityI*) *simp-all*
  then show *?thesis*
    using *assms execute-tm-write-many* **by** *simp*
qed

lemma *traces-tm-write-manyI*:
  assumes $0 \notin J$ **and** $\forall j{\in}J.\ j < k$ **and** $k \geq 2$ **and** $length\ tps = k$
    and $length\ tps' = k$
    and $\bigwedge j.\ j \in J \Longrightarrow tps'\ !\ j = tps\ !\ j\ |{:=}|\ h$
    and $\bigwedge j.\ j < k \Longrightarrow j \notin J \Longrightarrow tps'\ !\ j = tps\ !\ j$
    and $es = [(tps\ :\#:\ 0,\ tps\ :\#:\ 1)]$
  shows $traces\ (tm\text{-}write\text{-}many\ J\ h)\ tps\ es\ tps'$
proof

```
  show execute (tm-write-many J h) (0, tps) (length es) = (length (tm-write-many J h), tps')
    using execute-tm-write-manyI[OF assms(1−7)] tm-write-many-def assms(8) by simp
  show ⋀i. i < length es ⟹
        fst (execute (tm-write-many J h) (0, tps) i) < length (tm-write-many J h)
    using assms(8) tm-write-many-def by simp
  show ⋀i. i < length es ⟹
      snd (execute (tm-write-many J h) (0, tps) (Suc i)) :#: 0 = fst (es ! i) ∧
      snd (execute (tm-write-many J h) (0, tps) (Suc i)) :#: 1 = snd (es ! i)
    using execute-tm-write-manyI[OF assms(1−7)] tm-write-many-def assms(3,6,7,8)
    by (metis One-nat-def Suc-1 Suc-lessD fst-conv length-Cons less-Suc0
      less-eq-Suc-le list.size(3) nth-Cons-0 snd-conv)
qed

lemma traces-tm-write-repeat-1I:
  assumes 1 < length tps
    and es = map (λi. (tps :#: 0, tps :#: 1 + Suc i)) [0..<m]
    and tps' = tps[1 := overwrite (tps ! 1) h m]
  shows traces (tm-write-repeat 1 h m) tps es tps'
proof
  let ?M = tm-write-repeat 1 h m
  have length es = m
    using assms(2) by simp
  moreover have length ?M = m
    using tm-write-repeat-def by simp
  ultimately show execute ?M (0, tps) (length es) = (length ?M, tps')
    using assms by (simp add: execute-tm-write-repeat)
  show ⋀i. i < length es ⟹ fst (execute ?M (0, tps) i) < length ?M
    using assms execute-tm-write-repeat tm-write-repeat-def by simp
  show execute ?M (0, tps) (Suc i) <#> 0 = fst (es ! i) ∧
      execute ?M (0, tps) (Suc i) <#> 1 = snd (es ! i)
    if i < length es for i
  proof −
    have Suc i ≤ m
      using assms ‹length es = m› that by linarith
    then have execute ?M (0, tps) (Suc i) = (Suc i, tps[1 := overwrite (tps ! 1) h (Suc i)])
      using that execute-tm-write-repeat assms by blast
    then show ?thesis
      using overwrite-def assms(1,2) that by simp
  qed
qed
```

5.1.5 Memorizing in states

We need some results for the traces of "cartesian" machines used for the memorizing-in-states technique introduced in Section 2.5.

```
lemma cartesian-trace:
  assumes turing-machine (Suc k) G M
    and immobile M k (Suc k)
    and M' = cartesian M G
    and k ≥ 2
    and ∀i<length zs. zs ! i < G
    and trace M (start-config (Suc k) zs) es cfg
  shows trace M' (start-config k zs) es (squish G (length M) cfg)
proof (rule traceI')
  show execute M' (start-config k zs) (length es) = squish G (length M) cfg
    using assms cartesian-execute-start-config trace-def by auto
  have len: length M' = G ∗ length M
    by (simp add: assms(3) length-cartesian)
  have G > 0
    using assms(1) turing-machine-sequential-def by (simp add: turing-machine-def)
  show fst (execute M' (start-config k zs) i) < length M'
    if i < length es for i
  proof (rule ccontr)
```

```
    assume ¬ fst (execute M' (start-config k zs) i) < length M'
    then have fst (execute M' (start-config k zs) i) ≥ length M'
      by simp
    then have fst (execute M' (start-config k zs) i) = length M'
      using assms(1,3) cartesian-tm'
      by (metis (no-types, lifting) Suc-1 Suc-le-D assms(4) start-config-def start-config-length
        le-add2 le-add-same-cancel2 le-antisym less-Suc-eq-0-disj prod.sel(1) turing-machine-execute-states)
    then have fst (squish G (length M) (execute M (start-config (Suc k) zs) i)) = G * length M
      using assms cartesian-execute-start-config len by simp
    moreover have fst (execute M (start-config (Suc k) zs) i) ≤ length M
      using assms(1) assms(6) that trace-def by auto
    ultimately have fst (execute M (start-config (Suc k) zs) i) = length M
      using squish-halt-state ‹0 < G› by simp
    then show False
      using that assms(6) trace-def by auto
  qed
  show execute M' (start-config k zs) (Suc i) <#> 0 = fst (es ! i) ∧
       execute M' (start-config k zs) (Suc i) <#> 1 = snd (es ! i)
    if i < length es for i
  proof (rule ccontr)
    assume a: ¬ (snd (execute M' (start-config k zs) (Suc i)) :#: 0 = fst (es ! i) ∧
      snd (execute M' (start-config k zs) (Suc i)) :#: 1 = snd (es ! i))
    have *: execute M' (start-config k zs) (Suc i) =
       squish G (length M) (execute M (start-config (Suc k) zs) (Suc i))
      using assms cartesian-execute-start-config by blast
    then have execute M' (start-config k zs) (Suc i) <#> 0 =
       squish G (length M) (execute M (start-config (Suc k) zs) (Suc i)) <#> 0
      by simp
    also have ... = execute M (start-config (Suc k) zs) (Suc i) <#> 0
      using squish-head-pos assms execute-num-tapes start-config-length le-imp-less-Suc zero-less-Suc
      by presburger
    also have ... = fst (es ! i)
      using that assms trace-def by simp
    finally have fst: execute M' (start-config k zs) (Suc i) <#> 0 = fst (es ! i) .

    from * have execute M' (start-config k zs) (Suc i) <#> 1 =
       squish G (length M) (execute M (start-config (Suc k) zs) (Suc i)) <#> 1
      by simp
    also have ... = execute M (start-config (Suc k) zs) (Suc i) <#> 1
      using squish-head-pos assms execute-num-tapes start-config-length le-imp-less-Suc zero-less-Suc
      by presburger
    also have ... = snd (es ! i)
      using that assms trace-def by simp
    finally have execute M' (start-config k zs) (Suc i) <#> 1 = snd (es ! i) .
    then show False
      using fst a by simp
  qed
qed

lemma cartesian-traces:
  assumes turing-machine (Suc k) G M
    and immobile M k (Suc k)
    and M' = cartesian M G
    and k ≥ 2
    and ∀ i<length zs. zs ! i < G
    and traces M (snd (start-config (Suc k) zs)) es tps
  shows traces M' (snd (start-config k zs)) es (butlast tps)
proof -
  have trace M (start-config (Suc k) zs) es (length M, tps)
    using assms(6) traces-def by (simp add: start-config-def)
  then have trace M' (start-config k zs) es (squish G (length M) (length M, tps))
    using assms cartesian-trace by simp
  then show ?thesis
```

using *squish traces-def* **by** (*simp add*: *assms*(*3*) *start-config-def length-cartesian*)
qed

lemma *traces-tapes-length*:
assumes *turing-machine k G M*
and *length tps* = *k*
and *traces M tps es tps'*
shows *length tps'* = *k*
using *assms traces-def execute-num-tapes* **by** (*metis snd-conv trace-def*)

lemma *icartesian*:
assumes *turing-machine* (*k* + *2*) *G M*
and $\bigwedge j.\ j < k \Longrightarrow$ *immobile M* (*j* + *2*) (*k* + *2*)
and $\forall$ *i*<*length zs*. *zs* ! *i* < *G*
and *traces M* (*snd* (*start-config* (*k* + *2*) *zs*)) *es tps*
shows *traces* (*icartesian k M G*) (*snd* (*start-config 2 zs*)) *es* (*take 2 tps*)
using *assms*(*1*,*2*,*4*)
proof (*induction k arbitrary*: *M tps*)
case *0*
let *?M* = *icartesian 0 M G*
have ||*start-config* (*0* + *2*) *zs*|| = *2*
using *0 start-config-length* **by** *simp*
then have *length tps* = *2*
using *0 traces-tapes-length* **by** (*metis One-nat-def Suc-1 add-2-eq-Suc'*)
then have *tps* = *take 2 tps*
by *simp*
then have *traces ?M* (*snd* (*start-config 2 zs*)) *es* (*take 2 tps*)
using *0* **by** (*metis icartesian.simps*(*1*) *plus-nat.add-0*)
then show *?case*
by *auto*
next
case (*Suc k*)
let *?M* = *cartesian M G*
have *turing-machine* (*Suc* (*k* + *2*)) *G M*
using *Suc* **by** *simp*
moreover have *immobile M* (*k* + *2*) (*Suc* (*k* + *2*))
using *Suc* **by** *simp*
moreover have *k* + *2* $\geq$ *2*
by *simp*
moreover have *traces M* (*snd* (*start-config* (*Suc* (*k* + *2*)) *zs*)) *es tps*
using *Suc* **by** *simp*
ultimately have ∗: *traces ?M* (*snd* (*start-config* (*k* + *2*) *zs*)) *es* (*butlast tps*)
using *assms*(*3*) *cartesian-traces* **by** *simp*

have *turing-machine* (*k* + *2*) *G ?M*
using ‹*2* ≤ *k* + *2*› ‹*turing-machine* (*Suc* (*k* + *2*)) *G M*› *cartesian-tm'* **by** *blast*
moreover have $\bigwedge j.\ j < k \Longrightarrow$ *immobile ?M* (*j* + *2*) (*k* + *2*)
using *cartesian-immobile Suc* **by** *simp*
ultimately have *traces* (*icartesian k ?M G*) (*snd* (*start-config 2 zs*)) *es* (*take 2* (*butlast tps*))
using ∗ *Suc* **by** *simp*
moreover have *take 2* (*butlast tps*) = *take 2 tps*
proof −
have *length tps* = *Suc k* + *2*
using *start-config-length traces-tapes-length Suc*
by (*metis* (*mono-tags*, *lifting*) *add-gr-0 zero-less-Suc*)
then show *?thesis*
by (*simp add*: *take-butlast*)
qed
ultimately show *?case*
by *simp*
qed

end

5.2 Constructing polynomials in polynomial time

theory *Oblivious-Polynomial*
  imports *Oblivious*
begin

Our current goal is to simulate a polynomial time multi-tape TM on a two-tape oblivious TM in polynomial time. To help with the obliviousness we first want to mark on the simulator's output tape a space that is at least as large as the space the simulated TM uses on any of its tapes but that still is only polynomial in size. In this section we construct oblivious Turing machines for that.

An upper bound for the size this space is provided by the simulated TM's running time, which by assumption is polynomially bounded. So for our purposes any polynomially bounded function that bounds the running time will do.

In this section we devise a family of two-tape oblivious TMs that contains for every polynomial degree $d \geq 1$ a TM that writes $\mathbf{1}^{p(n)}$ to the output tape for some polynomial p with $p(n) \geq n^d$, where n is the length of the input to the TM. Together with a TM that appends a constant number c of ones we get a family of TMs, parameterized by c and d, that runs in polynomial time and writes more than $c + n^d$ symbols **1** to the work tape.

This meets our goal for this section because every polynomially bounded function is bounded by a function of the form $n \mapsto c + n^d$ for some $c, d \in \mathbb{N}$.

The TMs in the family are built from three building block TMs. The first TM initializes its output tape with $\mathbf{1}^n$ where n is the length of the input. The second TM multiplies the number of symbols on the output tape by (roughly) the length of the input, turning a sequence $\mathbf{1}^m$ into (roughly) $\mathbf{1}^{mn}$ for arbitrary m. The third TM appends $\mathbf{1}^c$ for some constant c. By repeating the second TM we can achieve arbitrarily high polynomial degrees.

All three TMs do essentially only one thing with the input, namely copying it to the output tape while mapping all symbols to **1**, which is an operation that depends only on the length of the input. Therefore all three TMs are oblivious, and combining them also yields an oblivious TM.

The Turing machines require one extra symbol beyond the four default symbols, but work for all alphabet sizes $G \geq 5$.

locale *turing-machine-poly* =
  fixes *G* :: *nat*
  assumes *G*: *G* ≥ *5*
begin

lemma *G-ge4* [*simp*]: *G* ≥ *4*
  using *G* **by** *linarith*

abbreviation *tps-ones* :: *symbol list* ⇒ *nat* ⇒ *tape list* **where**
  tps-ones zs m ≡
    [(⌊*zs*⌋, *1*),
     (λ*i*. *if i* = *0 then* ▷ *else if i* ≤ *m then* **1** *else* □, *1*)]

5.2.1 Initializing the output tape

The first building block is a TM that "copies" the input to the output tape, thereby replacing every symbol by the symbol **1**.

definition *tmA* :: *machine* **where**
  tmA ≡
    tm-right 0 ;; *tm-right 1* ;; *tm-const-until 0 1* {□} **1** ;; *tm-cr 0* ;; *tm-cr 1*

lemma *tmA-tm*: *turing-machine 2 G tmA*
  unfolding *tmA-def* **using** *tm-right-tm tm-const-until-tm tm-cr-tm G* **by** *simp*

definition *tm1* ≡ *tm-right 0*
definition *tm2* ≡ *tm1* ;; *tm-right 1*
definition *tm3* ≡ *tm2* ;; *tm-const-until 0 1* {□} **1**
definition *tm4* ≡ *tm3* ;; *tm-cr 0*

definition *tm5* ≡ *tm4* ;; *tm-cr 1*

lemma *tm5-eq-tmA*: *tm5* = *tmA*
  unfolding *tmA-def tm5-def tm4-def tm3-def tm2-def tm1-def* **by** *simp*

definition *tps0* :: *symbol list* ⇒ *tape list* **where**
  tps0 zs ≡
    [(⌊*zs*⌋, *0*),
     (λ*i*. *if i* = *0 then* ▷ *else* □, *0*)]

lemma *length-tps0* [*simp*]: *length* (*tps0 n*) = *2*
  unfolding *tps0-def* **by** *simp*

definition *tps1* :: *symbol list* ⇒ *tape list* **where**
  tps1 zs ≡
    [(⌊*zs*⌋, *1*),
     (λ*i*. *if i* = *0 then* ▷ *else* □, *0*)]

definition *es1* :: (*nat* × *nat*) *list* **where**
  es1 ≡ [(*1*, *0*)]

lemma *tm1*: *traces tm1* (*tps0 zs*) *es1* (*tps1 zs*)
  unfolding *tm1-def*
  by (*rule traces-tm-right-0I*) (*simp-all add*: *tps0-def tps1-def es1-def*)

definition *tps2* :: *symbol list* ⇒ *tape list* **where**
  tps2 zs ≡
    [(⌊*zs*⌋, *1*),
     (λ*i*. *if i* = *0 then* ▷ *else* □, *1*)]

definition *es2* :: (*nat* × *nat*) *list* **where**
  es2 ≡ *es1* @ [(*1*, *1*)]

lemma *length-es2*: *length es2* = *2*
  unfolding *es1-def es2-def* **by** *simp*

lemma *tm2*: *traces tm2* (*tps0 zs*) *es2* (*tps2 zs*)
  unfolding *tm2-def es2-def*
proof (*rule traces-sequential*[*OF tm1*])
  show *traces* (*tm-right 1*) (*tps1 zs*) [(*1*, *1*)] (*tps2 zs*)
    using *tps1-def tps2-def* **by** (*intro traces-tm-right-1I*) *simp-all*
qed

definition *tps3* :: *symbol list* ⇒ *tape list* **where**
  tps3 zs ≡
    [(⌊*zs*⌋, *length zs* + *1*),
     (λ*i*. *if i* = *0 then* ▷ *else if i* ≤ *length zs then* **1** *else* □, *length zs* + *1*)]

definition *es23* :: *nat* ⇒ (*nat* × *nat*) *list* **where**
  es23 n ≡ *map* (λ*i*. (*i* + *2*, *i* + *2*)) [*0*..<*n*] @ [(*n* + *1*, *n* + *1*)]

definition *es3* :: *nat* ⇒ (*nat* × *nat*) *list* **where**
  es3 n ≡ *es2* @ (*es23 n*)

lemma *length-es3*: *length* (*es3 n*) = *n* + *3*
  unfolding *es3-def es23-def* **using** *length-es2* **by** *simp*

lemma *tm3*:
  assumes *proper-symbols zs*
  shows *traces tm3* (*tps0 zs*) (*es3* (*length zs*)) (*tps3 zs*)
  unfolding *tm3-def es3-def*
proof (*rule traces-sequential*[*OF tm2*])
  show *traces* (*tm-const-until 0 1* {□} **1**) (*tps2 zs*) (*es23* (*length zs*)) (*tps3 zs*)

proof (*rule traces-tm-const-until-01I*)
  show *1* < *length* (*tps2 zs*)
    using *tps2-def* **by** *simp*
  show *rneigh* (*tps2 zs* ! *0*) {□} (*length zs*)
    using *rneigh-def contents-def tps2-def assms* **by** *auto*
  show *es23* (*length zs*) =
      map (λ*i*. (*tps2 zs* :#: *0* + *Suc i*, *tps2 zs* :#: *1* + *Suc i*))
      [*0*..<*length zs*] @
      [(*tps2 zs* :#: *0* + *length zs*, *tps2 zs* :#: *1* + *length zs*)]
    unfolding *es23-def* **using** *tps2-def* **by** *simp*
  show *tps3 zs* = (*tps2 zs*)
      [*0* := *tps2 zs* ! *0* |+| *length zs*,
       1 := *constplant* (*tps2 zs* ! *1*) **1** (*length zs*)]
    using *tps3-def tps2-def constplant* **by** *auto*
qed
qed

definition *tps4* :: *symbol list* ⇒ *tape list* **where**
  tps4 zs ≡
   [(⌊*zs*⌋, *1*),
    (λ*i*. *if i* = *0 then* ▷ *else if i* ≤ *length zs then* **1** *else* □, *length zs* + *1*)]

definition *es34* :: *nat* ⇒ (*nat* × *nat*) *list* **where**
  es34 n ≡ *map* (λ*i*. (*i*, *n* + *1*)) (*rev* [*0*..<*n* + *1*]) @ [(*0*, *n* + *1*)] @ [(*1*, *n* + *1*)]

definition *es4* :: *nat* ⇒ (*nat* × *nat*) *list* **where**
  es4 n ≡ *es3 n* @ *es34 n*

lemma *length-es4*: *length* (*es4 n*) = *2* ∗ *n* + *6*
  unfolding *es4-def es34-def* **using** *length-es3* **by** *simp*

lemma *tm4*:
  assumes *proper-symbols zs*
  shows *traces tm4* (*tps0 zs*) (*es4* (*length zs*)) (*tps4 zs*)
  unfolding *tm4-def es4-def*
proof (*rule traces-sequential*[*OF tm3*])
  show *proper-symbols zs*
    using *assms* **.**
  show *traces* (*tm-cr 0*) (*tps3 zs*) (*es34* (*length zs*)) (*tps4 zs*)
  proof (*rule traces-tm-cr-0I*)
    show *1* < *length* (*tps3 zs*)
      using *tps3-def* **by** *simp*
    show *clean-tape* (*tps3 zs* ! *0*)
      using *assms tps3-def* **by** *simp*
    show *es34* (*length zs*) =
        map (λ*i*. (*i*, *tps3 zs* :#: *1*)) (*rev* [*0*..<*tps3 zs* :#: *0*]) @
        [(*0*, *tps3 zs* :#: *1*), (*1*, *tps3 zs* :#: *1*)]
      using *tps3-def es34-def* **by** *simp*
    show *tps4 zs* = (*tps3 zs*)[*0* := *tps3 zs* ! *0* |#=| *1*]
      using *tps3-def tps4-def* **by** *simp*
  qed
qed

definition *tps5* :: *symbol list* ⇒ *nat* ⇒ *tape list* **where**
  tps5 zs m ≡ *tps-ones zs m*

definition *es45* :: *nat* ⇒ (*nat* × *nat*) *list* **where**
  es45 n ≡ *map* (λ*i*. (*1*, *i*)) (*rev* [*0*..<*n* + *1*]) @ [(*1*, *0*)] @ [(*1*, *1*)]

definition *es5* :: *nat* ⇒ (*nat* × *nat*) *list* **where**
  es5 n ≡ *es4 n* @ *es45 n*

lemma *length-es5*: *length* (*es5 n*) = *3* ∗ *n* + *9*

unfolding *es5-def es45-def* **using** *length-es4* **by** *simp*

lemma *tm5*:
assumes *proper-symbols zs*
shows *traces tm5 (tps0 zs) (es5 (length zs)) (tps5 zs (length zs))*
unfolding *tm5-def es5-def*
proof (*rule traces-sequential*[*OF tm4*])
show *proper-symbols zs*
using *assms* **.**
show *traces (tm-cr 1) (tps4 zs) (es45 (length zs)) (tps5 zs (length zs))*
proof (*rule traces-tm-cr-1I*)
show $1 < $ *length (tps4 zs)*
using *tps4-def* **by** *simp*
show *clean-tape (tps4 zs ! 1)*
using *tps4-def clean-tape-def* **by** *simp*
show *es45 (length zs)* =
map (Pair (tps4 zs :#: 0)) (rev [0..<tps4 zs :#: 1]) @
[(tps4 zs :#: 0, 0), (tps4 zs :#: 0, 1)]
using *tps4-def* **by** (*simp add*: *es45-def*)
show *tps5 zs (length zs) = (tps4 zs)[1 := tps4 zs ! 1 |#=| 1]*
using *tps4-def tps5-def* **by** *simp*
qed
qed

corollary *tmA*:
assumes *proper-symbols zs*
shows *traces tmA (tps0 zs) (es5 (length zs)) (tps-ones zs (length zs))*
using *tps5-def tm5-eq-tmA tm5 assms* **by** *simp*

lemma *length-tps-ones* [*simp*]: *length (tps-ones zs m) = 2*
by *simp*

5.2.2 Multiplying by the input length

The next Turing machine turns a symbol sequence $\mathbf{1}^m$ on its output tape into $\mathbf{1}^{m+1+mn}$ where n is the length of the input.

The TM first appends to the output tape symbols a | symbol. Then it performs a loop with m iterations. In each iteration it replaces a **1** before the | by **0** in order to count the iterations. Then it copies (replacing each symbol by **1**) the input after the output tape symbols. After the loop the output tape contains $\mathbf{0}^m|\mathbf{1}^{mn}$. Finally the | and **0** symbols are replaced by **1** symbols, yielding the desired result.

definition *tmB* :: *machine* **where**
tmB ≡
tm-right-until 1 {□} ;;
tm-write 1 | ;;
tm-cr 1 ;;
WHILE tm-right-until 1 {**1**, |} ; *λrs. rs ! 1* = **1** *DO*
tm-write 1 **0** ;;
tm-right-until 1 {*0*} ;;
tm-const-until 0 1 {□} **1** ;;
tm-cr 0 ;;
tm-cr 1
DONE ;;
tm-write 1 **1** ;;
tm-cr 1 ;;
tm-const-until 1 1 {**1**} **1** ;;
tm-cr 1

lemma *tmB-tm*: *turing-machine 2 G tmB*
unfolding *tmB-def*
using *tm-right-until-tm tm-write-tm tm-cr-tm tm-const-until-tm G*
turing-machine-loop-turing-machine turing-machine-sequential-turing-machine
by *simp*

```
definition tmB1 ≡ tm-right-until 1 {□}
definition tmB2 ≡ tmB1 ;; tm-write 1 |
definition tmB3 ≡ tmB2 ;; tm-cr 1
definition tmK1 ≡ tm-right-until 1 {1, |}
definition tmL1 ≡ tm-write 1 0
definition tmL2 ≡ tmL1 ;; tm-right-until 1 {□}
definition tmL3 ≡ tmL2 ;; tm-const-until 0 1 {□} 1
definition tmL4 ≡ tmL3 ;; tm-cr 0
definition tmL5 ≡ tmL4 ;; tm-cr 1
definition tmLoop ≡ WHILE tmK1 ; λrs. rs ! 1 = 1 DO tmL5 DONE
definition tmB4 ≡ tmB3 ;; tmLoop
definition tmB5 ≡ tmB4 ;; tm-write 1 1
definition tmB6 ≡ tmB5 ;; tm-cr 1
definition tmB7 ≡ tmB6 ;; tm-const-until 1 1 {1} 1
definition tmB8 ≡ tmB7 ;; tm-cr 1

lemma tmB-eq-tmB8: tmB = tmB8
  unfolding tmB-def tmB1-def tmB2-def tmB3-def tmK1-def tmL1-def tmL2-def tmL3-def
    tmL4-def tmL5-def tmLoop-def tmB4-def tmB5-def tmB6-def tmB7-def tmB8-def
  by simp

definition tpsB1 :: symbol list ⇒ nat ⇒ tape list where
  tpsB1 zs m ≡
    [(⌊zs⌋, 1),
     (λi. if i = 0 then ▷ else if i ≤ m then 1 else □, m + 1)]

definition esB1 :: nat ⇒ nat ⇒ (nat × nat) list where
  esB1 n m ≡ map (λi. (1, 1 + Suc i)) [0..<m] @ [(1, 1 + m)]

lemma length-esB1: length (esB1 n m) = m + 1
  using esB1-def by (metis Suc-eq-plus1 length-append-singleton length-map length-upt minus-nat.diff-0)

lemma tmB1:
  assumes proper-symbols zs
  shows traces tmB1 (tps-ones zs m) (esB1 (length zs) m) (tpsB1 zs m)
  unfolding tmB1-def
proof (rule traces-tm-right-until-1I[where ?n=m])
  show 1 < length (tps-ones zs m)
    by simp
  show rneigh (tps-ones zs m ! 1) {0} m
    using rneighI by simp
  show esB1 (length zs) m =
      map (λi. (tps-ones zs m :#: 0, tps-ones zs m :#: 1 + Suc i)) [0..<m] @
      [(tps-ones zs m :#: 0, tps-ones zs m :#: 1 + m)]
    by (simp add: esB1-def)
  show tpsB1 zs m = (tps-ones zs m)[1 := tps-ones zs m ! 1 |+| m]
    using tpsB1-def by simp
qed

definition tpsB2 :: symbol list ⇒ nat ⇒ tape list where
  tpsB2 zs m ≡
    [(⌊zs⌋, 1),
     (λi. if i = 0 then ▷ else if i ≤ m then 1 else if i = m + 1 then | else □, m + 1)]

definition esB12 :: nat ⇒ nat ⇒ (nat × nat) list where
  esB12 n m ≡ [(1, m + 1)]

definition esB2 :: nat ⇒ nat ⇒ (nat × nat) list where
  esB2 n m ≡ esB1 n m @ esB12 n m

lemma length-esB2: length (esB2 n m) = m + 2
  by (simp add: esB12-def esB2-def length-esB1)
```

lemma *tmB2*:
  assumes *proper-symbols zs*
  shows *traces tmB2* (*tps-ones zs m*) (*esB2* (*length zs*) *m*) (*tpsB2 zs m*)
  unfolding *tmB2-def esB2-def*
proof (*rule traces-sequential*[*OF tmB1*])
  show *proper-symbols zs*
    using *assms* **.**
  show *traces* (*tm-write 1* |) (*tpsB1 zs m*) (*esB12* (*length zs*) *m*) (*tpsB2 zs m*)
  proof (*rule traces-tm-write-1I*)
    show $1 < length$ (*tpsB1 zs m*)
      using *tpsB1-def* **by** *simp-all*
    show *esB12* (*length zs*) $m = [(tpsB1\ zs\ m\ {:}\#{:}\ 0,\ tpsB1\ zs\ m\ {:}\#{:}\ 1)]$
      using *tpsB1-def* **by** (*simp add*: *esB12-def*)
    show *tpsB2 zs m* = (*tpsB1 zs m*)[*1* := *tpsB1 zs m* ! *1* |:=| |]
      using *tpsB2-def tpsB1-def* **by** *auto*
  qed
qed

definition *tpsB3* :: *symbol list* ⇒ *nat* ⇒ *tape list* **where**
  tpsB3 zs m ≡
    [(⌊*zs*⌋, *1*),
     ($\lambda i.$ *if* $i = 0$ *then* ▷ *else if* $i \leq m$ *then* **1** *else if* $i = m + 1$ *then* | *else 0*, *1*)]

definition *esB23* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
  esB23 n m ≡ *map* (*Pair 1*) (*rev* [*0*..<*m* + *1*]) @ [(*1*, *0*), (*1*, *1*)]

definition *esB3* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
  esB3 n m ≡ *esB2 n m* @ *esB23 n m*

lemma *length-esB3*: *length* (*esB3 n m*) $= 2 * m + 5$
  by (*simp add*: *esB3-def length-esB2 esB23-def*)

lemma *tmB3*:
  assumes *proper-symbols zs*
  shows *traces tmB3* (*tps-ones zs m*) (*esB3* (*length zs*) *m*) (*tpsB3 zs m*)
  unfolding *tmB3-def esB3-def*
proof (*rule traces-sequential*[*OF tmB2*])
  show *proper-symbols zs*
    using *assms* **.**
  show *traces* (*tm-cr 1*) (*tpsB2 zs m*) (*esB23* (*length zs*) *m*) (*tpsB3 zs m*)
  proof (*rule traces-tm-cr-1I*)
    show $1 < length$ (*tpsB2 zs m*)
      using *tpsB2-def* **by** *simp*
    show *clean-tape* (*tpsB2 zs m* ! *1*)
      using *tpsB2-def clean-tape-def* **by** *simp*
    show *esB23* (*length zs*) *m* =
        map (*Pair* (*tpsB2 zs m* :#: *0*)) (*rev* [*0*..<*tpsB2 zs m* :#: *1*]) @
        [(*tpsB2 zs m* :#: *0*, *0*), (*tpsB2 zs m* :#: *0*, *1*)]
      by (*simp add*: *esB23-def tpsB2-def*)
    show *tpsB3 zs m* = (*tpsB2 zs m*)[*1* := *tpsB2 zs m* ! *1* |#=| *1*]
      using *tpsB2-def tpsB3-def* **by** *simp*
  qed
qed

definition *tpsK0* :: *symbol list* ⇒ *nat* ⇒ *nat* ⇒ *tape list* **where**
  tpsK0 zs m i ≡
    [(⌊*zs*⌋, *1*),
     ($\lambda x.$ *if* $x = 0$ *then* ▷
          else if $x \leq i$ *then* **0**
          else if $x \leq m$ *then* **1**
          else if $x = m + 1$ *then* |
          else if $x \leq m + 1 + i * length\ zs$ *then* **1**

else 0,
1)]

lemma *tpsK0-eq-tpsB3*: *tpsK0 zs m 0 = tpsB3 zs m*
using *tpsK0-def tpsB3-def* **by** *auto*

definition *tpsK1* :: *symbol list* ⇒ *nat* ⇒ *nat* ⇒ *tape list* **where**
tpsK1 zs m i ≡
[(⌊*zs*⌋, *1*),
(λ*x*. *if x = 0 then* ▷
else if x ≤ *i then* **0**
else if x ≤ *m then* **1**
else if x = m + 1 then |
else if x ≤ *m + 1 + i* ∗ *length zs then* **1**
else 0,
i + 1)]

definition *esK1* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esK1 n m i ≡ *map* (λ*i*. (*1*, *1 + Suc i*)) [*0*..<*i*] @ [(*1*, *i + 1*)]

lemma *length-esK1*: *length* (*esK1 n m i*) = *i + 1*
by (*simp add*: *esK1-def*)

lemma *tmK1*:
assumes *proper-symbols zs* **and** *i* < *m*
shows *traces tmK1* (*tpsK0 zs m i*) (*esK1* (*length zs*) *m i*) (*tpsK1 zs m i*)
unfolding *tmK1-def*
proof (*rule traces-tm-right-until-1I*[**where** *?n=i*])
show *1* < *length* (*tpsK0 zs m i*)
using *tpsK0-def* **by** *simp*
show *rneigh* (*tpsK0 zs m i* ! *1*) {**1**, |} *i*
using *tpsK0-def rneigh-def assms*(*2*) **by** *simp*
show *esK1* (*length zs*) *m i* =
map (λ*j*. (*tpsK0 zs m i* :#: *0*, *tpsK0 zs m i* :#: *1 + Suc j*)) [*0*..<*i*] @
[(*tpsK0 zs m i* :#: *0*, *tpsK0 zs m i* :#: *1 + i*)]
by (*simp add*: *esK1-def tpsK0-def*)
show *tpsK1 zs m i* = (*tpsK0 zs m i*)[*1* := *tpsK0 zs m i* ! *1* |+| *i*]
by (*simp add*: *tpsK1-def tpsK0-def*)
qed

definition *tpsL1* :: *symbol list* ⇒ *nat* ⇒ *nat* ⇒ *tape list* **where**
tpsL1 zs m i ≡
[(⌊*zs*⌋, *1*),
(λ*x*. *if x = 0 then* ▷
else if x ≤ *i + 1 then* **0**
else if x ≤ *m then* **1**
else if x = m + 1 then |
else if x ≤ *m + 1 + i* ∗ *length zs then* **1**
else 0,
i + 1)]

definition *esL1* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esL1 n m i ≡ [(*1*, *i + 1*)]

lemma *tmL1*:
assumes *proper-symbols zs*
shows *traces tmL1* (*tpsK1 zs m i*) (*esL1* (*length zs*) *m i*) (*tpsL1 zs m i*)
unfolding *tmL1-def* **using** *G*
by (*intro traces-tm-write-1I*) (*auto simp add*: *tpsL1-def tpsK1-def esL1-def*)

definition *tpsL2* :: *symbol list* ⇒ *nat* ⇒ *nat* ⇒ *tape list* **where**
tpsL2 zs m i ≡
[(⌊*zs*⌋, *1*),

$(\lambda x.\ \mathit{if}\ x = 0\ \mathit{then}\ \triangleright$
$\quad \mathit{else\ if}\ x \leq i + 1\ \mathit{then}\ \mathbf{0}$
$\quad \mathit{else\ if}\ x \leq m\ \mathit{then}\ \mathbf{1}$
$\quad \mathit{else\ if}\ x = m + 1\ \mathit{then}\ |$
$\quad \mathit{else\ if}\ x \leq m + 1 + i * \mathit{length}\ zs\ \mathit{then}\ \mathbf{1}$
$\quad \mathit{else}\ 0,$
$m + 2 + i * \mathit{length}\ zs)]$

definition *esL12* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esL12 n m i ≡
map (λ*j*. (*1*, *i* + *1* + *Suc j*)) [*0*..<*m* + *2* + *i* ∗ *n* − (*i* + *1*)] @
[(*1*, *i* + *1* + (*m* + *2* + *i* ∗ *n* − (*i* + *1*)))]

definition *esL2* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esL2 n m i ≡ *esL1 n m i* @ *esL12 n m i*

lemma *length-esL2*: *i* < *m* ⟹ *length* (*esL2 n m i*) = *3* + *m* − *i* + *i* ∗ *n*
by (*auto simp add*: *esL2-def esL12-def esL1-def*)

A simplified upper bound for the running time:

corollary *length-esL2′*: *i* < *m* ⟹ *length* (*esL2 n m i*) ≤ *3* + *m* + *i* ∗ *n*
by (*simp add*: *length-esL2*)

lemma *tmL2*:
assumes *proper-symbols zs* **and** *i* < *m*
shows *traces tmL2* (*tpsK1 zs m i*) (*esL2* (*length zs*) *m i*) (*tpsL2 zs m i*)
unfolding *tmL2-def esL2-def*
proof (*rule traces-sequential*[*OF tmL1*])
show *proper-symbols zs*
using *assms*(*1*) **.**
show *traces* (*tm-right-until 1* {*0*}) (*tpsL1 zs m i*) (*esL12* (*length zs*) *m i*) (*tpsL2 zs m i*)
proof (*rule traces-tm-right-until-1I*)
show *1* < *length* (*tpsL1 zs m i*)
using *tpsL1-def* **by** *simp*
show *rneigh* (*tpsL1 zs m i* ! *1*) {*0*} (*m* + *2* + *i* ∗ *length zs* − (*i* + *1*))
using *rneigh-def assms*(*2*) **by** (*auto simp add*: *tpsL1-def*)
show *esL12* (*length zs*) *m i* =
map (λ*j*. (*tpsL1 zs m i* :#: *0*, *tpsL1 zs m i* :#: *1* + *Suc j*))
[*0*..<*m* + *2* + *i* ∗ *length zs* − (*i* + *1*)] @
[(*tpsL1 zs m i* :#: *0*,
tpsL1 zs m i :#: *1* + (*m* + *2* + *i* ∗ *length zs* − (*i* + *1*)))]
using *assms*(*2*) **by** (*simp add*: *tpsL1-def esL12-def*)
show *tpsL2 zs m i* = (*tpsL1 zs m i*) [*1* := *tpsL1 zs m i* ! *1* |+| *m* + *2* + *i* ∗ *length zs* − (*i* + *1*)]
using *assms*(*2*) **by** (*simp add*: *tpsL1-def tpsL2-def*)
qed
qed

definition *tpsL3* :: *symbol list* ⇒ *nat* ⇒ *nat* ⇒ *tape list* **where**
tpsL3 zs m i ≡
[(⌊*zs*⌋, *length zs* + *1*),
$(\lambda x.\ \mathit{if}\ x = 0\ \mathit{then}\ \triangleright$
$\quad \mathit{else\ if}\ x \leq i + 1\ \mathit{then}\ \mathbf{0}$
$\quad \mathit{else\ if}\ x \leq m\ \mathit{then}\ \mathbf{1}$
$\quad \mathit{else\ if}\ x = m + 1\ \mathit{then}\ |$
$\quad \mathit{else\ if}\ x \leq m + 1 + (i + 1) * \mathit{length}\ zs\ \mathit{then}\ \mathbf{1}$
$\quad \mathit{else}\ 0,$
$m + 2 + (i + 1) * \mathit{length}\ zs)]$

definition *esL23* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esL23 n m i ≡
map (λ*j*. (*1* + *Suc j*, *m* + *2* + *i* ∗ *n* + *Suc j*)) [*0*..<*n*] @ [(*1* + *n*, *m* + *2* + *i* ∗ *n* + *n*)]

definition *esL3* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**

esL3 n m i ≡ *esL2 n m i* @ *esL23 n m i*

lemma *length-esL3*: $i < m \Longrightarrow length\ (esL3\ n\ m\ i) \leq 4 + m + (i + 1) * n$
by (*auto simp add*: *esL3-def esL23-def*) (*metis group-cancel.add1 length-esL2'*)

lemma *tmL3*:
assumes *proper-symbols zs* **and** $i < m$
shows *traces tmL3* (*tpsK1 zs m i*) (*esL3* (*length zs*) *m i*) (*tpsL3 zs m i*)
unfolding *tmL3-def esL3-def*
proof (*rule traces-sequential*[*OF tmL2*])
show *proper-symbols zs* **and** $i < m$
using *assms* **.**
show *traces* (*tm-const-until 0 1* {□} **1**) (*tpsL2 zs m i*) (*esL23* (*length zs*) *m i*) (*tpsL3 zs m i*)
proof (*rule traces-tm-const-until-01I*)
show $1 <$ *length* (*tpsL2 zs m i*)
using *tpsL2-def* **by** *simp*
show *rneigh* (*tpsL2 zs m i* ! *0*) {*0*} (*length zs*)
using *assms*(*1*) *rneigh-def contents-def* **by** (*auto simp add*: *tpsL2-def*)
show *esL23* (*length zs*) *m i* =
map (λ*j*. (*tpsL2 zs m i* :#: *0* + *Suc j*, *tpsL2 zs m i* :#: *1* + *Suc j*)) [*0*..<*length zs*] @
[(*tpsL2 zs m i* :#: *0* + *length zs*, *tpsL2 zs m i* :#: *1* + *length zs*)]
using *assms* **by** (*simp add*: *esL23-def tpsL2-def*)
show *tpsL3 zs m i* = (*tpsL2 zs m i*)
[*0* := *tpsL2 zs m i* ! *0* |+| *length zs*,
1 := *constplant* (*tpsL2 zs m i* ! *1*) **1** (*length zs*)]
using *constplant assms* **by** (*auto simp add*: *tpsL2-def tpsL3-def*)
qed
qed

definition *tpsL4* :: *symbol list* ⇒ *nat* ⇒ *nat* ⇒ *tape list* **where**
tpsL4 zs m i ≡
[(⌊*zs*⌋, *1*),
(λ*x*. *if* $x = 0$ *then* ▷
else if $x \leq i + 1$ *then* **0**
else if $x \leq m$ *then* **1**
else if $x = m + 1$ *then* |
else if $x \leq m + 1 + (i + 1) * length\ zs$ *then* **1**
else *0*,
$m + 2 + (i + 1) * length\ zs$)]

definition *esL34* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esL34 n m i ≡
map (λ*j*. (*j*, $m + 2 + (i + 1) * n$)) (*rev* [*0*..<$n + 1$]) @ [(*0*, $m + 2 + (i + 1) * n$), (*1*, $m + 2 + (i + 1) * n$)]

lemma *length-esL34*: $i < m \Longrightarrow length\ (esL34\ n\ m\ i) = n + 3$
unfolding *esL34-def* **by** *simp*

definition *esL4* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esL4 n m i ≡ *esL3 n m i* @ *esL34 n m i*

lemma *length-esL4*: $i < m \Longrightarrow length\ (esL4\ n\ m\ i) \leq 7 + m + (i + 2) * n$
using *length-esL3 length-esL34*
by (*auto simp add*: *esL4-def*) (*metis ab-semigroup-add-class.add-ac*(*1*) *group-cancel.add2*)

lemma *tmL4*:
assumes *proper-symbols zs* **and** $i < m$
shows *traces tmL4* (*tpsK1 zs m i*) (*esL4* (*length zs*) *m i*) (*tpsL4 zs m i*)
unfolding *tmL4-def esL4-def*
proof (*rule traces-sequential*[*OF tmL3*])
show *proper-symbols zs* $i < m$
using *assms* **.**
show *traces* (*tm-cr 0*) (*tpsL3 zs m i*) (*esL34* (*length zs*) *m i*) (*tpsL4 zs m i*)

proof (*rule traces-tm-cr-0I*)
  show *1* < *length* (*tpsL3 zs m i*)
    using *tpsL3-def* **by** *simp*
  show *clean-tape* (*tpsL3 zs m i* ! *0*)
    using *tpsL3-def assms*(*1*) **by** *simp*
  show *esL34* (*length zs*) *m i* =
      map (λ*j*. (*j*, *tpsL3 zs m i* :#: *1*)) (*rev* [*0*..<*tpsL3 zs m i* :#: *0*]) @
      [(*0*, *tpsL3 zs m i* :#: *1*), (*1*, *tpsL3 zs m i* :#: *1*)]
    by (*simp add*: *esL34-def tpsL3-def*)
  show *tpsL4 zs m i* = (*tpsL3 zs m i*)[*0* := *tpsL3 zs m i* ! *0* |#=| *1*]
    by (*simp add*: *tpsL4-def tpsL3-def*)
qed
qed

definition *tpsL5* :: *symbol list* ⇒ *nat* ⇒ *nat* ⇒ *tape list* **where**
tpsL5 zs m i ≡
  [(⌊*zs*⌋, *1*),
   (λ*x*. *if x* = *0 then* ▷
        else if x ≤ *i* + *1 then* **0**
        else if x ≤ *m then* **1**
        else if x = *m* + *1 then* |
        else if x ≤ *m* + *1* + (*i* + *1*) ∗ *length zs then* **1**
        else 0,
    1)]

definition *esL45* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esL45 n m i ≡ *map* (*Pair 1*) (*rev* [*0*..<*m* + *2* + (*i* + *1*) ∗ *n*]) @ [(*1*, *0*), (*1*, *1*)]

definition *esL5* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esL5 n m i ≡ *esL4 n m i* @ *esL45 n m i*

lemma *length-esL5*: *i* < *m* ⟹ *length* (*esL5 n m i*) ≤ *11* + *2* ∗ *m* + (*2* ∗ *i* + *3*) ∗ *n*
proof −
  assume *a*: *i* < *m*
  have *length* (*esL5 n m i*) = *length* (*esL4 n m i*) + *length* (*esL45 n m i*)
    using *esL5-def* **by** *simp*
  also have ... ≤ *7* + *m* + (*i* + *2*) ∗ *n* + *length* (*esL45 n m i*)
    using *length-esL4*[*OF a*] **by** *simp*
  also have ... = *7* + *m* + (*i* + *2*) ∗ *n* + (*m* + *2* + (*i* + *1*) ∗ *n* + *2*)
    using *esL45-def* **by** *simp*
  also have ... = *11* + *2* ∗ *m* + (*2* ∗ *i* + *3*) ∗ *n*
    by *algebra*
  finally show *?thesis* **.**
qed

lemma *tmL5*:
  assumes *proper-symbols zs* **and** *i* < *m*
  shows *traces tmL5* (*tpsK1 zs m i*) (*esL5* (*length zs*) *m i*) (*tpsL5 zs m i*)
  unfolding *tmL5-def esL5-def*
proof (*rule traces-sequential*[*OF tmL4*])
  show *proper-symbols zs i* < *m*
    using *assms* **.**
  show *traces* (*tm-cr 1*) (*tpsL4 zs m i*) (*esL45* (*length zs*) *m i*) (*tpsL5 zs m i*)
  proof (*rule traces-tm-cr-1I*)
    show *1* < *length* (*tpsL4 zs m i*)
      using *tpsL4-def* **by** *simp*
    show *clean-tape* (*tpsL4 zs m i* ! *1*)
      using *tpsL4-def assms clean-tapeI* **by** *simp*
    show *esL45* (*length zs*) *m i* =
        map (*Pair* (*tpsL4 zs m i* :#: *0*)) (*rev* [*0*..<*tpsL4 zs m i* :#: *1*]) @
        [(*tpsL4 zs m i* :#: *0*, *0*), (*tpsL4 zs m i* :#: *0*, *1*)]
      by (*simp add*: *esL45-def tpsL4-def*)
    show *tpsL5 zs m i* = (*tpsL4 zs m i*)[*1* := *tpsL4 zs m i* ! *1* |#=| *1*]

by (*simp add*: *tpsL4-def tpsL5-def*)
qed
qed

definition *esLoop-do* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esLoop-do n m i ≡ *esK1 n m i* @ [(*1*, *i* + *1*)] @ *esL5 n m i* @ [(*1*, *1*)]

Using $i < m$ we get this upper bound for the running time of an iteration independent of i.

lemma *length-esLoop-do*: *i* < *m* ⟹ *length* (*esLoop-do n m i*) ≤ *14* + *3* ∗ *m* + (*2* ∗ *m* + *3*) ∗ *n*
proof −
assume *i* < *m*
have *length* (*esLoop-do n m i*) = *length* (*esK1 n m i*) + *length* (*esL5 n m i*) + *2*
unfolding *esLoop-do-def* **by** *simp*
also have ... = *i* + *3* + (*length* (*esL5 n m i*))
using *length-esK1* **by** *simp*
also have ... ≤ *i* + *14* + *2* ∗ *m* + (*2* ∗ *i* + *3*) ∗ *n*
using *length-esL5*[*OF* ‹*i* < *m*›] **by** (*simp add*: *add.assoc*)
also have ... ≤ *14* + *3* ∗ *m* + (*2* ∗ *i* + *3*) ∗ *n*
using ‹*i* < *m*› **by** *simp*
also have ... ≤ *14* + *3* ∗ *m* + (*2* ∗ *m* + *3*) ∗ *n*
using ‹*i* < *m*› **by** *simp*
finally show *?thesis* **.**
qed

lemma *tmLoop-do*:
assumes *proper-symbols zs* **and** *i* < *m*
shows *trace tmLoop* (*0*, *tpsK0 zs m i*) (*esLoop-do* (*length zs*) *m i*) (*0*, *tpsL5 zs m i*)
unfolding *tmLoop-def*
proof (*rule tm-loop-sem-true-tracesI*[*OF tmK1 tmL5*])
show *proper-symbols zs proper-symbols zs i* < *m i* < *m*
using *assms* **by** *simp-all*
show *read* (*tpsK1 zs m i*) ! *1* = **1**
using *tpsK1-def assms*(*2*) *read-def* **by** *simp*
show *esLoop-do* (*length zs*) *m i* =
esK1 (*length zs*) *m i* @
[(*tpsK1 zs m i* :#: *0*, *tpsK1 zs m i* :#: *1*)] @
esL5 (*length zs*) *m i* @ [(*tpsL5 zs m i* :#: *0*, *tpsL5 zs m i* :#: *1*)]
by (*simp add*: *esLoop-do-def esK1-def tpsK1-def esL5-def tpsL5-def*)
qed

lemma *tpsL5-eq-tpsK0*:
assumes *proper-symbols zs* **and** *i* < *m*
shows *tpsL5 zs m i* = *tpsK0 zs m* (*Suc i*)
using *assms tpsL5-def tpsK0-def* **by** *auto*

lemma *tmLoop-iteration*:
assumes *proper-symbols zs* **and** *i* < *m*
shows *trace tmLoop* (*0*, *tpsK0 zs m i*) (*esLoop-do* (*length zs*) *m i*) (*0*, *tpsK0 zs m* (*Suc i*))
using *assms tmLoop-do tpsL5-eq-tpsK0* **by** *simp*

definition *esLoop-done* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esLoop-done n m ≡ *concat* (*map* (*esLoop-do n m*) [*0*..<*m*])

lemma *tmLoop-done*:
assumes *proper-symbols zs*
shows *trace tmLoop* (*0*, *tpsK0 zs m 0*) (*esLoop-done* (*length zs*) *m*) (*0*, *tpsK0 zs m m*)
using *assms tm-loop-trace-simple* **by** (*simp add*: *tmLoop-iteration esLoop-done-def*)

lemma *length-esLoop-done*: *length* (*esLoop-done n m*) ≤ *m* ∗ (*14* + *3* ∗ *m* + (*2* ∗ *m* + *3*) ∗ *n*)
using *length-concat-le*[*OF length-esLoop-do*] *esLoop-done-def* **by** *simp*

definition *tpsK-break* :: *symbol list* ⇒ *nat* ⇒ *tape list* **where**
tpsK-break zs m ≡

[(⌊zs⌋, 1),
 (λx. if x = 0 then ▷
  else if x ≤ m then **0**
  else if x = m + 1 then |
  else if x ≤ m + 1 + m * length zs then **1**
  else 0,
 m + 1)]

definition *esK-break* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esK-break n m ≡ *map* (λ*i*. (*1*, *1* + *Suc i*)) [*0*..<*m*] @ [(*1*, *1* + *m*)]

lemma *length-esK-break*: *length* (*esK-break n m*) = *m* + *1*
unfolding *esK-break-def* **by** *simp*

lemma *tmK1-break*:
assumes *proper-symbols zs*
shows *traces tmK1* (*tpsK0 zs m m*) (*esK-break* (*length zs*) *m*) (*tpsK-break zs m*)
unfolding *tmK1-def*
proof (*rule traces-tm-right-until-1I*[**where** *?n*=*m*])
show *1* < *length* (*tpsK0 zs m m*)
using *tpsK0-def* **by** *simp*
show *rneigh* (*tpsK0 zs m m* ! *1*) {**1**, |} *m*
using *rneigh-def* **by** (*simp add*: *tpsK0-def*)
show *esK-break* (*length zs*) *m* =
map (λ*j*. (*tpsK0 zs m m* :#: *0*, *tpsK0 zs m m* :#: *1* + *Suc j*)) [*0*..<*m*] @
[(*tpsK0 zs m m* :#: *0*, *tpsK0 zs m m* :#: *1* + *m*)]
by (*simp add*: *esK-break-def tpsK0-def*)
show *tpsK-break zs m* = (*tpsK0 zs m m*)[*1* := *tpsK0 zs m m* ! *1* |+| *m*]
by (*auto simp add*: *tpsK-break-def tpsK0-def*)
qed

definition *esLoop-break* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esLoop-break n m ≡ *esK-break n m* @ [(*1*, *m* + *1*)]

lemma *length-esLoop-break*: *length* (*esLoop-break n m*) = *m* + *2*
unfolding *esLoop-break-def* **using** *length-esK-break* **by** *simp*

lemma *tmLoop-break*:
assumes *proper-symbols zs*
shows *traces tmLoop* (*tpsK0 zs m m*) (*esLoop-break* (*length zs*) *m*) (*tpsK-break zs m*)
unfolding *tmLoop-def esLoop-break-def*
proof (*rule tm-loop-sem-false-traces*[*OF tmK1-break*])
show *proper-symbols zs*
using *assms* **.**
show *read* (*tpsK-break zs m*) ! *1* ≠ **1**
using *tpsK-break-def read-def* **by** *simp*
show *esK-break* (*length zs*) *m* @ [(*1*, *m* + *1*)] =
esK-break (*length zs*) *m* @ [(*tpsK-break zs m* :#: *0*, *tpsK-break zs m* :#: *1*)]
by (*simp add*: *esK-break-def tpsK-break-def*)
qed

definition *esLoop* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
esLoop n m ≡ *esLoop-done n m* @ *esLoop-break n m*

lemma *length-esLoop*: *length* (*esLoop n m*) ≤ *m* * (*14* + *3* * *m* + (*2* * *m* + *3*) * *n*) + *m* + *2*
unfolding *esLoop-def* **using** *length-esLoop-done* **by** (*simp add*: *length-esLoop-break*)

lemma *length-esLoop'*: *length* (*esLoop n m*) ≤ *2* + *18* * *m* * *m* + *5* * *m* * *m* * *n*
proof −
have *length* (*esLoop n m*) ≤ *m* * (*14* + *3* * *m* + (*2* * *m* + *3*) * *n*) + *m* + *2*
using *length-esLoop* **.**
also have ... = *14* * *m* + *3* * *m* * *m* + (*2* * *m* * *m* + *3* * *m*) * *n* + *m* + *2*
by *algebra*

also have ... $\leq$ *15 * m * m + 3 * m * m + (2 * m * m + 3 * m) * n + 2*
by *simp*
also have ... $\leq$ *2 + 18 * m * m + 5 * m * m * n*
by *simp*
finally show *?thesis* **.**
qed

lemma *tmLoop*:
assumes *proper-symbols zs*
shows *traces tmLoop (tpsK0 zs m 0) (esLoop (length zs) m) (tpsK-break zs m)*
unfolding *esLoop-def* **using** *assms* **by** *(intro traces-additive[OF tmLoop-done tmLoop-break])*

lemma *tmLoop′*:
assumes *proper-symbols zs*
shows *traces tmLoop (tpsB3 zs m) (esLoop (length zs) m) (tpsK-break zs m)*
using *assms tmLoop tpsK0-eq-tpsB3* **by** *simp*

definition *esB4* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ *(nat* $\times$ *nat) list* **where**
esB4 n m $\equiv$ *esB3 n m @ esLoop n m*

lemma *length-esB4*: *length (esB4 n m)* $\leq$ *7 + 20 * m * m + 5 * m * m * n*
proof −
have *length (esB4 n m)* $\leq$ *2 * m + 5 + m * (14 + 3 * m + (2 * m + 3) * n) + m + 2*
unfolding *esB4-def*
using *length-esB3 length-esLoop*
by *(smt (verit) ab-semigroup-add-class.add-ac(1) add-less-cancel-left le-eq-less-or-eq length-append)*
also have ... = *2 * m + 5 + (14 * m + 3 * m * m + (2 * m * m + 3 * m) * n) + m + 2*
by *algebra*
also have ... = *7 + 17 * m + 3 * m * m + (2 * m * m + 3 * m) * n*
by *simp*
also have ... $\leq$ *7 + 17 * m + 3 * m * m + 5 * m * m * n*
by *simp*
also have ... $\leq$ *7 + 20 * m * m + 5 * m * m * n*
by *simp*
finally show *?thesis* **.**
qed

lemma *tmB4*:
assumes *proper-symbols zs*
shows *traces tmB4 (tps-ones zs m) (esB4 (length zs) m) (tpsK-break zs m)*
unfolding *tmB4-def esB4-def*
using *assms* **by** *(intro traces-sequential[OF tmB3 tmLoop′])*

definition *tpsB5* :: *symbol list* $\Rightarrow$ *nat* $\Rightarrow$ *tape list* **where**
tpsB5 zs m $\equiv$
[(⌊zs⌋, 1),
(λx. if x = 0 then ▹
else if x $\leq$ *m then* **0**
else if x $\leq$ *m + 1 + m * length zs then* **1**
else 0,
m + 1)]

definition *esB5* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ *(nat* $\times$ *nat) list* **where**
esB5 n m $\equiv$ *esB4 n m @ [(1, m + 1)]*

lemma *length-esB5*: *length (esB5 n m)* $\leq$ *8 + 20 * m * m + 5 * m * m * n*
unfolding *esB5-def*
using *length-esB4*
by *(metis Suc-le-mono length-append-singleton one-plus-numeral plus-1-eq-Suc plus-nat.simps(2)*
semiring-norm(2) semiring-norm(4) semiring-norm(8))

lemma *tmB5*:
assumes *proper-symbols zs*

```
  shows traces tmB5 (tps-ones zs m) (esB5 (length zs) m) (tpsB5 zs m)
  unfolding tmB5-def esB5-def
proof (rule traces-sequential[OF tmB4])
  show proper-symbols zs
    using assms .
  show traces (tm-write 1 1) (tpsK-break zs m) [(1, m + 1)] (tpsB5 zs m)
  proof (rule traces-tm-write-1I)
    show 1 < length (tpsK-break zs m)
      using tpsK-break-def by simp-all
    show [(1, m + 1)] = [(tpsK-break zs m :#: 0, tpsK-break zs m :#: 1)]
      using tpsK-break-def by simp
    show tpsB5 zs m = (tpsK-break zs m)[1 := tpsK-break zs m ! 1 |:=| 1]
      by (auto simp add: tpsK-break-def tpsB5-def)
  qed
qed

definition tpsB6 :: symbol list ⇒ nat ⇒ tape list where
  tpsB6 zs m ≡
    [(⌊zs⌋, 1),
     (λx. if x = 0 then ▷
          else if x ≤ m then 0
          else if x ≤ m + 1 + m ∗ length zs then 1
          else 0,
      1)]

definition esB56 :: nat ⇒ nat ⇒ (nat × nat) list where
  esB56 n m ≡ map (Pair 1) (rev [0..<m + 1]) @ [(1, 0), (1, 1)]

definition esB6 :: nat ⇒ nat ⇒ (nat × nat) list where
  esB6 n m ≡ esB5 n m @ esB56 n m

lemma length-esB6: length (esB6 n m) ≤ 11 + 21 ∗ m ∗ m + 5 ∗ m ∗ m ∗ n
proof −
  have length (esB6 n m) ≤ 8 + 20 ∗ m ∗ m + 5 ∗ m ∗ m ∗ n + m + 3
    unfolding esB6-def esB56-def using length-esB5 by (simp add: ab-semigroup-add-class.add-ac(1))
  also have ... = 11 + 20 ∗ m ∗ m + 5 ∗ m ∗ m ∗ n + m
    by simp
  also have ... ≤ 11 + 21 ∗ m ∗ m + 5 ∗ m ∗ m ∗ n
    by simp
  finally show ?thesis .
qed

lemma tmB6:
  assumes proper-symbols zs
  shows traces tmB6 (tps-ones zs m) (esB6 (length zs) m) (tpsB6 zs m)
  unfolding tmB6-def esB6-def
proof (rule traces-sequential[OF tmB5])
  show proper-symbols zs
    using assms .
  show traces (tm-cr 1) (tpsB5 zs m) (esB56 (length zs) m) (tpsB6 zs m)
  proof (rule traces-tm-cr-1I)
    show 1 < length (tpsB5 zs m)
      using tpsB5-def by simp
    show clean-tape (tpsB5 zs m ! 1)
      using tpsB5-def clean-tape-def by simp
    show esB56 (length zs) m =
        map (Pair (tpsB5 zs m :#: 0)) (rev [0..<tpsB5 zs m :#: 1]) @
        [(tpsB5 zs m :#: 0, 0), (tpsB5 zs m :#: 0, 1)]
      by (simp add: esB56-def tpsB5-def)
    show tpsB6 zs m = (tpsB5 zs m)[1 := tpsB5 zs m ! 1 |#=| 1]
      by (simp add: tpsB6-def tpsB5-def)
  qed
qed
```

definition *tpsB7* :: *symbol list* ⇒ *nat* ⇒ *tape list* **where**
  tpsB7 zs m ≡
    [(⌊*zs*⌋, *1*),
     (λ*x*. *if x* = *0 then* ▷
          else if x ≤ *m* + *1* + *m* ∗ *length zs then* **1**
          else 0,
      m + *1*)]

definition *esB67* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
  esB67 n m ≡ *map* (λ*i*. (*1*, *1* + *Suc i*)) [*0*..<*m*] @ [(*1*, *1* + *m*)]

definition *esB7* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
  esB7 n m ≡ *esB6 n m* @ *esB67 n m*

lemma *length-esB7*: *length* (*esB7 n m*) ≤ *12* + *22* ∗ *m* ∗ *m* + *5* ∗ *m* ∗ *m* ∗ *n*
proof −
  have *length* (*esB7 n m*) ≤ *11* + *21* ∗ *m* ∗ *m* + *5* ∗ *m* ∗ *m* ∗ *n* + *m* + *1*
    unfolding *esB7-def esB67-def*
    using *length-esB6*
    by (*smt* (*verit*) *add.commute add-Suc-right add-le-cancel-right length-append length-append-singleton*
      length-map length-upt minus-nat.diff-0 plus-1-eq-Suc)
  also have ... ≤ *12* + *22* ∗ *m* ∗ *m* + *5* ∗ *m* ∗ *m* ∗ *n*
    by *simp*
  finally show *?thesis* **.**
qed

lemma *tmB7*:
  assumes *proper-symbols zs*
  shows *traces tmB7* (*tps-ones zs m*) (*esB7* (*length zs*) *m*) (*tpsB7 zs m*)
  unfolding *tmB7-def esB7-def*
proof (*rule traces-sequential*[*OF tmB6*])
  show *proper-symbols zs*
    using *assms* **.**
  show *traces* (*tm-const-until 1 1* {**1**} **1**) (*tpsB6 zs m*) (*esB67* (*length zs*) *m*) (*tpsB7 zs m*)
  proof (*rule traces-tm-const-until-11I*)
    show *1* < *length* (*tpsB6 zs m*) *3* < *G*
      using *tpsB6-def G G-ge4* **by** *simp-all*
    show *rneigh* (*tpsB6 zs m* ! *1*) {**1**} *m*
      using *tpsB6-def* **by** (*intro rneighI*) *auto*
    show *esB67* (*length zs*) *m* =
        map (λ*i*. (*tpsB6 zs m* :#: *0*, *tpsB6 zs m* :#: *1* + *Suc i*)) [*0*..<*m*] @
        [(*tpsB6 zs m* :#: *0*, *tpsB6 zs m* :#: *1* + *m*)]
      by (*simp add*: *tpsB6-def esB67-def*)
    show *tpsB7 zs m* = (*tpsB6 zs m*) [*1* := *constplant* (*tpsB6 zs m* ! *1*) **1** *m*]
      using *constplant* **by** (*auto simp add*: *tpsB7-def tpsB6-def*)
  qed
qed

definition *tpsB8* :: *symbol list* ⇒ *nat* ⇒ *tape list* **where**
  tpsB8 zs m ≡
    [(⌊*zs*⌋, *1*),
     (λ*x*. *if x* = *0 then* ▷
          else if x ≤ *m* + *1* + *m* ∗ *length zs then* **1**
          else 0,
      1)]

definition *esB78* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
  esB78 n m ≡ *map* (*Pair 1*) (*rev* [*0*..<*m* + *1*]) @ [(*1*, *0*), (*1*, *1*)]

definition *esB8* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
  esB8 n m ≡ *esB7 n m* @ *esB78 n m*

```
lemma length-esB8: length (esB8 n m) ≤ 15 + 23 ∗ m ∗ m + 5 ∗ m ∗ m ∗ n
proof −
  have length (esB8 n m) ≤ 12 + 22 ∗ m ∗ m + 5 ∗ m ∗ m ∗ n + m + 3
    unfolding esB8-def esB78-def using length-esB7 by (simp add: ab-semigroup-add-class.add-ac(1))
  also have ... ≤ 15 + 23 ∗ m ∗ m + 5 ∗ m ∗ m ∗ n
    by simp
  finally show ?thesis .
qed

lemma tmB8:
  assumes proper-symbols zs
  shows traces tmB8 (tps-ones zs m) (esB8 (length zs) m) (tpsB8 zs m)
  unfolding tmB8-def esB8-def
proof (rule traces-sequential[OF tmB7])
  show proper-symbols zs
    using assms .
  show traces (tm-cr 1) (tpsB7 zs m) (esB78 (length zs) m) (tpsB8 zs m)
  proof (rule traces-tm-cr-1I)
    show 1 < length (tpsB7 zs m)
      using tpsB7-def by simp
    show clean-tape (tpsB7 zs m ! 1)
      using tpsB7-def clean-tape-def by simp
    show esB78 (length zs) m =
        map (Pair (tpsB7 zs m :#: 0)) (rev [0..<tpsB7 zs m :#: 1]) @
        [(tpsB7 zs m :#: 0, 0), (tpsB7 zs m :#: 0, 1)]
      by (simp add: esB78-def tpsB7-def)
    show tpsB8 zs m = (tpsB7 zs m)[1 := tpsB7 zs m ! 1 |#=| 1]
      by (simp add: tpsB8-def tpsB7-def)
  qed
qed

lemma tps-ones-eq-tpsB8: tpsB8 zs m = tps-ones zs (1 + m ∗ (length zs + 1))
  using tpsB8-def by auto

lemma tmB:
  assumes proper-symbols zs
  shows traces tmB (tps-ones zs m) (esB8 (length zs) m) (tps-ones zs (1 + m ∗ (length zs + 1)))
  using assms tps-ones-eq-tpsB8 tmB8 tmB-eq-tmB8 by simp
```

5.2.3 Appending a fixed number of symbols

The next Turing machine appends a constant number c of **1** symbols to the output tape.

```
definition tmC :: nat ⇒ machine where
  tmC c ≡
    tm-right-until 1 {□} ;;
    tm-write-repeat 1 𝟏 c ;;
    tm-cr 1

lemma tmC-tm: turing-machine 2 G (tmC c)
  unfolding tmC-def using tm-right-until-tm tm-write-repeat-tm tm-cr-tm G
  by simp

definition tmC1 ≡ tm-right-until 1 {□}
definition tmC2 c ≡ tmC1 ;; tm-write-repeat 1 𝟏 c
definition tmC3 c ≡ tmC2 c ;; tm-cr 1

definition tpsC1 :: symbol list ⇒ nat ⇒ tape list where
  tpsC1 zs m ≡
    [(⌊zs⌋, 1),
     (λx. if x = 0 then ▹
           else if x ≤ m then 𝟏
           else 0,
      m + 1)]
```

```
definition esC1 :: nat ⇒ nat ⇒ (nat × nat) list where
  esC1 n m ≡ map (λi. (1, 1 + Suc i)) [0..<m] @ [(1, 1 + m)]

lemma length-esC1: length (esC1 n m) = m + 1
  unfolding esC1-def by simp

lemma tmC1:
  assumes proper-symbols zs
  shows traces tmC1 (tps5 zs m) (esC1 (length zs) m) (tpsC1 zs m)
  unfolding tmC1-def
proof (rule traces-tm-right-until-1I[where ?n=m])
  show 1 < length (tps5 zs m)
    using tps5-def by simp
  show rneigh (tps5 zs m ! 1) {0} m
    using rneigh-def tps5-def by simp
  show esC1 (length zs) m =
      map (λi. (tps5 zs m :#: 0, tps5 zs m :#: 1 + Suc i)) [0..<m] @
      [(tps5 zs m :#: 0, tps5 zs m :#: 1 + m)]
    by (simp add: tps5-def esC1-def)
  show tpsC1 zs m = (tps5 zs m)[1 := tps5 zs m ! 1 |+| m]
    by (simp add: tps5-def tpsC1-def)
qed

definition tpsC2 :: symbol list ⇒ nat ⇒ nat ⇒ tape list where
  tpsC2 zs m c ≡
    [(⌊zs⌋, 1),
     (λx. if x = 0 then ▷
          else if x ≤ m + c then 𝟏
          else 0,
      m + 1 + c)]

definition esC12 :: nat ⇒ nat ⇒ nat ⇒ (nat × nat) list where
  esC12 n m c ≡ map (λi. (1, m + 1 + Suc i)) [0..<c]

definition esC2 :: nat ⇒ nat ⇒ nat ⇒ (nat × nat) list where
  esC2 n m c ≡ esC1 n m @ esC12 n m c

lemma length-esC2: length (esC2 n m c) = m + 1 + c
  unfolding esC2-def by (simp add: length-esC1 esC12-def)

lemma tmC2:
  assumes proper-symbols zs
  shows traces (tmC2 c) (tps5 zs m) (esC2 (length zs) m c) (tpsC2 zs m c)
  unfolding tmC2-def esC2-def
proof (rule traces-sequential[OF tmC1])
  show proper-symbols zs
    using assms .
  show traces (tm-write-repeat 1 𝟏 c) (tpsC1 zs m) (esC12 (length zs) m c) (tpsC2 zs m c)
  proof (rule traces-tm-write-repeat-1I)
    show 1 < length (tpsC1 zs m)
      using tpsC1-def by simp
    show esC12 (length zs) m c =
        map (λi. (tpsC1 zs m :#: 0, tpsC1 zs m :#: 1 + Suc i)) [0..<c]
      by (simp add: esC12-def tpsC1-def)
    show tpsC2 zs m c = (tpsC1 zs m)[1 := overwrite (tpsC1 zs m ! 1) 𝟏 c]
      by (auto simp add: tpsC2-def tpsC1-def overwrite-def)
  qed
qed

definition tpsC3 :: symbol list ⇒ nat ⇒ nat ⇒ tape list where
  tpsC3 zs m c ≡
    [(⌊zs⌋, 1),
```

```
    (λx. if x = 0 then ▷
         else if x ≤ m + c then 1
         else 0,
     1)]

definition esC23 :: nat ⇒ nat ⇒ nat ⇒ (nat × nat) list where
  esC23 n m c ≡ map (Pair 1) (rev [0..<m + 1 + c]) @ [(1, 0), (1, 1)]

definition esC3 :: nat ⇒ nat ⇒ nat ⇒ (nat × nat) list where
  esC3 n m c ≡ esC2 n m c @ esC23 n m c

lemma length-esC3: length (esC3 n m c) = 2 ∗ m + 2 ∗ c + 4
  unfolding esC3-def by (simp add: length-esC2 esC23-def)

lemma tmC3:
  assumes proper-symbols zs
  shows traces (tmC3 c) (tps5 zs m) (esC3 (length zs) m c) (tpsC3 zs m c)
  unfolding tmC3-def esC3-def
proof (rule traces-sequential[OF tmC2])
  show proper-symbols zs
    using assms .
  show traces (tm-cr 1) (tpsC2 zs m c) (esC23 (length zs) m c) (tpsC3 zs m c)
  proof (rule traces-tm-cr-1I)
    show 1 < length (tpsC2 zs m c)
      using tpsC2-def by simp
    show clean-tape (tpsC2 zs m c ! 1)
      using tpsC2-def clean-tape-def by simp
    show esC23 (length zs) m c =
        map (Pair (tpsC2 zs m c :#: 0)) (rev [0..<tpsC2 zs m c :#: 1]) @
        [(tpsC2 zs m c :#: 0, 0), (tpsC2 zs m c :#: 0, 1)]
      by (simp add: tpsC2-def esC23-def)
    show tpsC3 zs m c = (tpsC2 zs m c)[1 := tpsC2 zs m c ! 1 |#=| 1]
      by (simp add: tpsC2-def tpsC3-def)
  qed
qed

lemma tpsC3-eq-tps5: tpsC3 zs m c = tps5 zs (m + c)
  by (simp add: tpsC3-def tps5-def)

lemma tmC3-eq-tmC: tmC3 = tmC
  unfolding tmC-def tmC3-def tmC2-def tmC1-def by simp

lemma tmC:
  assumes proper-symbols zs
  shows traces (tmC c) (tps-ones zs m) (esC3 (length zs) m c) (tps-ones zs (m + c))
  using tmC3[OF assms] tmC3-eq-tmC tpsC3-eq-tps5 tps5-def by simp
```

5.2.4 Polynomials of higher degree

In order to construct polynomials of arbitrary degree, we repeat the TM *tmB*.

```
fun tm-degree :: nat ⇒ machine where
  tm-degree 0 = [] |
  tm-degree (Suc d) = tm-degree d ;; tmB

lemma tm-degree-tm: turing-machine 2 G (tm-degree d)
proof (induction d)
  case 0
  then show ?case
    by (simp add: turing-machine-def)
next
  case (Suc d)
  then show ?case
    using turing-machine-def tmB-tm
```

by (*metis tm-degree.simps*(*2*) *turing-machine-sequential-turing-machine*)
qed

The number of **1** symbols the TM *tm-degree d* outputs on an input of length n:

fun *m-degree* :: *nat* ⇒ *nat* ⇒ *nat* **where**
m-degree n 0 = *n* |
m-degree n (*Suc d*) = *1* + *m-degree n d* ∗ (*n* + *1*)

fun *es-degree* :: *nat* ⇒ *nat* ⇒ (*nat* ∗ *nat*) *list* **where**
es-degree n 0 = [] |
es-degree n (*Suc d*) = *es-degree n d* @ *esB8 n* (*m-degree n d*)

lemma *tm-degree*:
assumes *proper-symbols zs*
shows *traces*
(*tm-degree d*)
(*tps-ones zs* (*length zs*))
(*es-degree* (*length zs*) *d*)
(*tps-ones zs* (*m-degree* (*length zs*) *d*))
proof (*induction d*)
case *0*
then show *?case*
by *fastforce*
next
case (*Suc d*)
have *traces* (*tm-degree d* ;; *tmB*)
(*tps-ones zs* (*length zs*))
(*es-degree* (*length zs*) *d* @ *esB8* (*length zs*) (*m-degree* (*length zs*) *d*))
(*tps-ones zs* (*m-degree* (*length zs*) (*Suc d*)))
proof (*rule traces-sequential*[*OF Suc.IH*])
show *traces tmB* (*tps-ones zs* (*m-degree* (*length zs*) *d*))
(*esB8* (*length zs*) (*m-degree* (*length zs*) *d*))
(*tps-ones zs* (*m-degree* (*length zs*) (*Suc d*)))
using *tmB*[*OF assms*, *of m-degree* (*length zs*) *d*] *m-degree.simps*(*2*) **by** *presburger*
qed
then show *?case*
by *simp*
qed

A lower bound for the number of **1** symbols the TM *tm-degree d* outputs:

lemma *m-degree-ge-pow*: *m-degree n d* ≥ *n* ^ (*Suc d*)
proof (*induction d*)
case *0*
then show *?case*
by *simp*
next
case (*Suc d*)
have *m-degree n* (*Suc d*) = *1* + *m-degree n d* ∗ (*n* + *1*)
by *simp*
then have *m-degree n* (*Suc d*) ≥ *1* + *n* ^ *Suc d* ∗ (*n* + *1*)
using *Suc* **by** (*simp add*: *add-mono-thms-linordered-semiring*(*1*))
then have *m-degree n* (*Suc d*) ≥ *1* + *n* ^ *Suc d* ∗ *n* + *n* ^ *Suc d*
by *simp*
then have *m-degree n* (*Suc d*) ≥ *1* + *n* ^ (*Suc* (*Suc d*)) + *n* ^ *Suc d*
by (*metis power-Suc2*)
then show *?case*
by *simp*
qed

An upper bound for the number of **1** symbols the TM *tm-degree d* outputs:

lemma *m-degree-poly*: *big-oh-poly* (λ*n*. *m-degree n d*)
proof (*induction d*)
case *0*

have (λn. *m-degree n 0*) = (λn. *n*)
by *simp*
then show *?case*
using *big-oh-poly-poly*[*of 1*] **by** *simp*
next
case (*Suc d*)
have *big-oh-poly* (λn. *n* + *1*)
using *big-oh-poly-sum*[*OF big-oh-poly-poly*[*of 1*] *big-oh-poly-const*[*of 1*]]
by *simp*
then have *big-oh-poly* (λn. *m-degree n d* ∗ (*n* + *1*))
using *big-oh-poly-prod*[*OF Suc*] **by** *blast*
then have *big-oh-poly* (λn. *1* + *m-degree n d* ∗ (*n* + *1*))
using *big-oh-poly-sum*[*OF big-oh-poly-const*[*of 1*]] **by** *simp*
then show *?case*
by *simp*
qed

corollary *m-degree-plus-const-poly*: *big-oh-poly* (λn. *m-degree n d* + *c*)
using *m-degree-poly big-oh-poly-sum big-oh-poly-const* **by** *simp*

lemma *length-es-degree*: *big-oh-poly* (λn. *length* (*es-degree n d*))
proof (*induction d*)
case *0*
then show *?case*
using *big-oh-poly-const* **by** *simp*
next
case (*Suc d*)
have *big-oh-poly* (λn. *length* (*esB8 n* (*m-degree n d*)))
proof −
let *?m* = λn. *m-degree n d*
have *big-oh-poly ?m*
using *m-degree-poly* **by** *simp*
then have *big-oh-poly* (λn. *15* + *23* ∗ *?m n* ∗ *?m n* + *5* ∗ *?m n* ∗ *?m n* ∗ *n*)
using *big-oh-poly-sum big-oh-poly-const big-oh-poly-prod big-oh-poly-poly*[*of 1*]
by *simp*
then show *?thesis*
using *length-esB8 big-oh-poly-le* **by** *simp*
qed
then show *?case*
using *Suc big-oh-poly-sum* **by** *simp*
qed

Putting together the TM *tmA*, the TM *tm-degree d* for some *d*, and the TM *tmC c* for some *c*, we get a family of TMs parameterized by *d* and *c*. These TMs construct all the polynomials we need.

definition *tm-poly* :: *nat* ⇒ *nat* ⇒ *machine* **where**
tm-poly d c ≡ *tmA* ;; (*tm-degree d* ;; *tmC c*)

lemma *tm-poly-tm*: *turing-machine 2 G* (*tm-poly d c*)
unfolding *tm-poly-def* **using** *tmA-tm tm-degree-tm tmC-tm* **by** *simp*

definition *es-poly* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*) *list* **where**
es-poly n d c ≡ *es5 n* @ *es-degree n d* @ *esC3 n* (*m-degree n d*) *c*

On an input of length *n* the Turing machine *tm-poly d c* outputs *m-degree n d* + *c* symbols **1**.

lemma *tm-poly*:
assumes *proper-symbols zs*
shows *traces*
(*tm-poly d c*)
(*tps0 zs*)
(*es-poly* (*length zs*) *d c*)
(*tps-ones zs* (*m-degree* (*length zs*) *d* + *c*))
unfolding *tm-poly-def es-poly-def*
using *assms traces-sequential*[*OF tmA*] *traces-sequential*[*OF tm-degree*] *tmC*

by *simp*

The Turing machines run in polynomial time because their traces have polynomial length:

lemma *length-es-poly*: *big-oh-poly* ($\lambda n.$ *length* (*es-poly n d c*))
proof −
  have *big-oh-poly* ($\lambda n.$ *length* (*es5 n*))
    using *length-es5 big-oh-poly-const big-oh-poly-prod big-oh-poly-sum big-oh-poly-poly*[*of 1*]
    by *simp*
  moreover have *big-oh-poly* ($\lambda n.$ *length* (*esC3 n* (*m-degree n d*) *c*))
  proof −
    have $*$: ($\lambda n.$ *length* (*esC3 n* (*m-degree n d*) *c*)) = ($\lambda n.$ $2 * $ (*m-degree n d*) $+ 2 * c + 4$)
      using *length-esC3* **by** *fast*
    have *big-oh-poly* ($\lambda n.$ $2 *$ (*m-degree n d*) $+ 2 * c + 4$)
      using *m-degree-poly big-oh-poly-const big-oh-poly-prod big-oh-poly-sum* **by** *simp*
    then show *?thesis*
      by (*simp add*: $*$)
  qed
  ultimately have *big-oh-poly* ($\lambda n.$ *length* (*es5 n*) + *length* (*es-degree n d*) + *length* (*esC3 n* (*m-degree n d*) *c*))
    using *length-es-degree big-oh-poly-sum* **by** *blast*
  then show *?thesis*
    by (*simp add*: *es-poly-def add.assoc*)
qed

The Turing machine *tm-poly d c* outputs *m-degree n c* + *c* many **1** symbols on an input of length n. Hence for every polynomially bounded function f there is such a Turing machine outputting at least $f(n)$ symbols **1**.

lemma *m-degree-plus-const*:
  assumes *big-oh-poly f*
  obtains *d c* **where** $\forall n.$ *f n* $\leq$ *m-degree n d* + *c*
proof −
  obtain *c m d* **where** *f*: $\forall n{>}m.$ *f n* $\leq c * n$ ^ *d*
    using *assms big-oh-poly* **by** *auto*
  let *?d* = *Suc d*
  let *?k* = *max c m*
  have *n* ^ *?d* $\geq c * n$ ^ *d* **if** *n* > *?k* **for** *n*
    using *that* **by** *simp*
  moreover have *f n* $\leq c * n$ ^ *d* **if** *n* > *?k* **for** *n*
    using *f that* **by** *simp*
  ultimately have *1*: *f n* $\leq$ *n* ^ *?d* **if** *n* > *?k* **for** *n*
    using *that* **using** *order-trans* **by** *blast*
  define c' **where** c' = *Max* {*f n*| *n*. *n* $\leq$ *?k*}
  moreover have *finite* {*f n*| *n*. *n* $\leq$ *?k*}
    by *simp*
  ultimately have $c' \geq$ *f n* **if** *n* $\leq$ *?k* **for** *n*
    using *that Max.bounded-iff* **by** *blast*
  then have *f n* $\leq$ *n* ^ *?d* + c' **if** *n* $\leq$ *?k* **for** *n*
    by (*simp add*: *that trans-le-add2*)
  moreover have *f n* $\leq$ *n* ^ *?d* + c' **if** *n* > *?k* **for** *n*
    using *that 1* **by** *fastforce*
  ultimately have *f n* $\leq$ *n* ^ *?d* + c' **for** *n*
    using *leI* **by** *blast*
  then have *f n* $\leq$ *m-degree n d* + c' **for** *n*
    using *m-degree-ge-pow* **by** (*meson le-diff-conv less-le-trans not-le*)
  then show *?thesis*
    using *that* **by** *auto*
qed

The Turing machines are oblivious.

lemma *tm-poly-oblivious*: *oblivious* (*tm-poly d c*)
proof −
  have *tm*: *turing-machine 2 G* (*tm-poly d c*)
    using *tm-poly-tm* **by** *simp*
  have *init*: (*0*, *tps0 zs*) = *start-config 2 zs* **for** *zs*

using *tps0-def start-config-def contents-def* **by** *auto*
{
fix *zs*
assume *bit-symbols zs*
then have *proper*: *proper-symbols zs*
by *auto*
define *tps* **where** *tps* = *tps-ones zs* (*m-degree* (*length zs*) *d* + *c*)
moreover define *e* **where** *e* = (λ*n*. *es-poly n d c*)
ultimately have *trace* (*tm-poly d c*) (*start-config 2 zs*) (*e* (*length zs*)) (*length* (*tm-poly d c*), *tps*)
using *tm-poly init proper* **by** (*simp add*: *traces-def*)
}
then show *?thesis*
using *tm oblivious-def* **by** *fast*
qed

end

definition *start-tapes-2* :: *symbol list* ⇒ *tape list* **where**
start-tapes-2 zs ≡
[(⌊*zs*⌋, *0*),
(λ*i*. *if i* = *0 then* ▷ *else* □, *0*)]

definition *one-tapes-2* :: *symbol list* ⇒ *nat* ⇒ *tape list* **where**
one-tapes-2 zs m ≡
[(⌊*zs*⌋, *1*),
(⌊*replicate m* **1**⌋, *1*)]

The main result of this chapter. For every polynomially bounded function g there is a polynomial-time two-tape oblivious Turing machine that outputs at least $g(n)$ symbols **1** for every input length n.

lemma *polystructor*:
assumes *big-oh-poly g* **and** *G* ≥ *5*
shows ∃ *M es f*.
turing-machine 2 G M ∧
big-oh-poly (λ*n*. *length* (*es n*)) ∧
big-oh-poly f ∧
(∀ *n*. *g n* ≤ *f n*) ∧
(∀ *zs*. *proper-symbols zs* ⟶ *traces M* (*start-tapes-2 zs*) (*es* (*length zs*)) (*one-tapes-2 zs* (*f* (*length zs*))))
proof −
interpret *turing-machine-poly G*
using *assms*(*2*) **by** (*simp add*: *turing-machine-poly-def*)
obtain *d c* **where** *dc*: ∀ *n*. *g n* ≤ *m-degree n d* + *c*
using *assms*(*1*) *m-degree-plus-const* **by** *auto*
define *M* **where** *M* = *tm-poly d c*
define *es* **where** *es* = (λ*n*. *es-poly n d c*)
define *f* **where** *f* = (λ*n*. *m-degree n d* + *c*)
have *turing-machine 2 G M*
using *M-def tm-poly-tm assms*(*2*) **by** *simp*
moreover have *big-oh-poly* (λ*n*. *length* (*es n*))
using *es-def length-es-poly* **by** *simp*
moreover have ∀ *n*. *g n* ≤ *f n*
using *f-def dc* **by** *simp*
moreover have *big-oh-poly f*
using *f-def m-degree-plus-const-poly* **by** *simp*
moreover have
∀ *zs*. *proper-symbols zs* ⟶ *traces M* (*start-tapes-2 zs*) (*es* (*length zs*)) (*one-tapes-2 zs* (*f* (*length zs*)))
proof (*standard*, *standard*)
fix *zs*
assume *zs*: *proper-symbols zs*
have *traces M* (*tps0 zs*) (*es* (*length zs*)) (*tps-ones zs* (*f* (*length zs*)))
unfolding *M-def es-def f-def* **using** *tm-poly*[*OF zs*, *of d c*] **by** *simp*
moreover have *tps0* = *start-tapes-2*
using *tps0-def start-tapes-2-def* **by** *presburger*
ultimately have *traces M* (*start-tapes-2 zs*) (*es* (*length zs*)) (*tps-ones zs* (*f* (*length zs*)))

by *simp*
moreover have *one-tapes-2* = *tps-ones*
using *one-tapes-2-def contents-def* **by** *fastforce*
ultimately show *traces M* (*start-tapes-2 zs*) (*es* (*length zs*)) (*one-tapes-2 zs* (*f* (*length zs*)))
by *metis*
qed
ultimately show *?thesis*
by *auto*
qed

end

5.3 Existence of two-tape oblivious Turing machines

theory *Oblivious-2-Tape*
imports *Oblivious-Polynomial NP*
begin

In this section we show that for every polynomial-time multi-tape Turing machine there is a polynomial-time two-tape oblivious Turing machine that computes the same function and halts with its output tape head in cell number 1.

Consider a k-tape Turing machine M with polynomially bounded running time T. We construct a two-tape oblivious Turing machine S simulating M as follows.

Lemma *polystructor* from the previous section provides us with a polynomial-time two-tape oblivious TM and a function $f(n) \geq T(n)$ such that the TM outputs $\mathbf{1}^{f(n)}$ for all inputs of length n.

Executing this TM is the first thing our simulator does. The $f(n)$ symbols **1** mark the space S is going to use. Every cell $i = 0, \ldots, f(n) - 1$ of this space is to store a symbol that encodes a $(2k + 2)$-tuple consisting of:

- k symbols from M's alphabet representing the contents of all the i-th cells on the k tapes of M;
- k flags (called "head flags") signalling which of the k tape heads of M is in cell i;
- a flag (called "counter flag") initialized with 0;
- a flag (called "start flag") signalling whether $i = 0$.

Together the counter flags are a unary counter from 0 to $f(n)$. They are toggled from left to right. The start flags never change. The symbols and the head flags represent the k tapes of M at some step of the execution. By choice of f the TM M cannot use more than $f(n)$ cells on any tape. So the space marked with **1** symbols on the simulator's output tape suffices.

Next the simulator initializes the space of **1** symbols with code symbols representing the start configuration of M for the input given to the simulator.

Then the main loop of the simulation performs $f(n)$ iterations. In each iteration S performs one step of M's computation. In order to do this it performs several left-to-right and right-to-left sweeps over all the $f(n)$ cells in the formatted section of the output tape. A sweep will move the output tape head one cell right (respectively left) in each step. In this way the simulator's head positions at any time will only depend on $f(n)$, and hence on n. Thus the machine will be oblivious. Moreover $f(n) \geq T(n)$, and so M will be in the halting state after $f(n)$ iterations of the simulation. Counting the iterations to $f(n)$ is achieved via the counter flags.

Finally the simulator extracts from each code symbol the symbol corresponding to M's output tape, thus reconstructing the output of M on the simulator's output tape. Thanks to the start flags, the simulator can easily locate the leftmost cell and put the output tape head one to the right of it, as required.

The construction heavily uses the memorization-in-states technique (see Chapter 2.5). At first the machine features $2k + 1$ memorization tapes in addition to the input tape and output tape. The purpose of those tapes is to store M's state and the symbols under the tape heads of M in the currently simulated step of M's execution, as well as the k symbols to be written write and head movements to be executed by the simulator.

The next predicate expresses that a Turing machine halts within a time bound depending on the length of the input. We did not have a need for this fairly basic predicate yet, because so far we were always interested in the halting configuration, too, and so the predicate *computes-in-time* sufficed.

definition *time-bound* :: *machine* ⇒ *nat* ⇒ (*nat* ⇒ *nat*) ⇒ *bool* **where**
  time-bound M k T ≡
    ∀ *zs*. *bit-symbols zs* ⟶ *fst* (*execute M* (*start-config k zs*) (*T* (*length zs*))) = *length M*

lemma *time-bound-ge*:
  assumes *time-bound M k T* **and** ∀ *n*. *fmt n* ≥ *T n*
  shows *time-bound M k fmt*
  using *time-bound-def assms* **by** (*metis execute-after-halting-ge*)

The time bound also bounds the position of all the tape heads.

lemma *head-pos-le-time-bound*:
  assumes *turing-machine k G M*
    and *time-bound M k T*
    and *bit-symbols zs*
    and *j* < *k*
  shows *execute M* (*start-config k zs*) *t* <#> *j* ≤ *T* (*length zs*)
  using *assms time-bound-def*[*of M k T*] *execute-after-halting-ge head-pos-le-time*[*OF assms*(*1*,*4*)]
  by (*metis* (*no-types*, *lifting*) *nat-le-linear*)

The entire construction will take place in a locale that assumes a polynomial-time k-tape Turing machine M.

locale *two-tape* =
  fixes *M* :: *machine* **and** *k G* :: *nat* **and** *T* :: *nat* ⇒ *nat*
  assumes *tm-M*: *turing-machine k G M*
    and *poly-T*: *big-oh-poly T*
    and *time-bound-T*: *time-bound M k T*
begin

lemma *k-ge-2*: *k* ≥ *2*
  using *tm-M turing-machine-def* **by** *simp*

lemma *G-ge-4*: *G* ≥ *4*
  using *tm-M turing-machine-def* **by** *simp*

The construction of the simulator relies on the formatted space on the output tape to be large enough to hold the input. The size of the formatted space depends on the time bound T, which might be less than the length of the input. To ensure that the formatted space is large enough we increase the time bound while keeping it polynomial. The new bound is T':

definition *T'* :: *nat* ⇒ *nat* **where**
  T' n ≡ *T n* + *n*

lemma *poly-T'*: *big-oh-poly T'*
proof −
  have *big-oh-poly* (λ*n*. *n*)
    using *big-oh-poly-poly*[*of 1*] **by** *simp*
  then show *?thesis*
    using *T'-def big-oh-poly-sum*[*OF poly-T*] **by** *presburger*
qed

lemma *time-bound-T'*: *time-bound M k T'*
  using *T'-def time-bound-ge*[*OF time-bound-T*, *of T'*] **by** *simp*

5.3.1 Encoding multiple tapes into one

The symbols on the output tape of the simulator are supposed to encode a $(2k+2)$-tuple, where the first k elements assume one of the values in $\{0, \ldots, G-1\}$, whereas the other $k+2$ are flags with two possible values only. For uniformity we define an encoding where all elements range over G values and that works for tuples of every length.

fun *encode* :: *nat list* ⇒ *nat* **where**
  encode [] = *0* |
  encode (*x* # *xs*) = *x* + *G* ∗ *encode xs*

For every $m \in \mathbb{N}$, the enocoding is a bijective mapping from $\{0, \ldots, G-1\}^m$ to $\{0, \ldots, G^m - 1\}$.

```
lemma encode-surj:
  assumes n < G ^ m
  shows ∃xs. length xs = m ∧ (∀x∈set xs. x < G) ∧ encode xs = n
  using assms
proof (induction m arbitrary: n)
  case 0
  then show ?case
    by simp
next
  case (Suc m)
  define q where q = n div G
  define r where r = n mod G
  have q < G ^ m
    using Suc q-def
    by (auto simp add: less-mult-imp-div-less power-commutes)
  then obtain xs' where xs': length xs' = m ∀x∈set xs'. x < G encode xs' = q
    using Suc by auto
  have r < G
    using r-def G-ge-4 by simp
  have encode (r # xs') = n
    using xs'(3) q-def r-def G-ge-4 by simp
  moreover have ∀x∈set (r # xs'). x < G
    using xs'(2) ‹r < G› by simp
  moreover have length (r # xs') = Suc m
    using xs'(1) by simp
  ultimately show ?case
    by blast
qed

lemma encode-inj:
  assumes ∀x∈set xs. x < G
    and length xs = m
    and ∀y∈set ys. y < G
    and length ys = m
    and encode xs = encode ys
  shows xs = ys
  using assms
proof (induction m arbitrary: xs ys)
  case 0
  then show ?case
    by simp
next
  case (Suc m)
  obtain x xs' y ys' where xy: xs = x # xs' ys = y # ys'
    using Suc by (metis Suc-length-conv)
  then have len: length xs' = m length ys' = m
    using Suc by simp-all
  have ∗: x + G ∗ encode xs' = y + G ∗ encode ys'
    using Suc xy by simp
  moreover have (x + G ∗ encode xs') mod G = x
    by (simp add: Suc.prems(1) xy(1))
  moreover have (y + G ∗ encode ys') mod G = y
    by (simp add: Suc.prems(3) xy(2))
  ultimately have x = y
    by simp
  with ∗ have G ∗ encode xs' = G ∗ encode ys'
    by simp
  then have encode xs' = encode ys'
```

using *G-ge-4* **by** *simp*
with *len Suc xy* **have** $xs' = ys'$
by *simp*
then show *?case*
using ‹$x = y$› *xy* **by** *simp*
qed

lemma *encode-less*:
assumes $\forall x \in set\ xs.\ x < G$
shows $encode\ xs < G\ \hat{}\ (length\ xs)$
using *assms*
proof (*induction xs*)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons x xs*)
then have $x < G$
by *simp*
have $encode\ (x\ \#\ xs) = x + G * encode\ xs$
by *simp*
also have $... \le x + G * (G\ \hat{}\ (length\ xs) - 1)$
using *Cons* **by** *simp*
also have $... = x + G * G\ \hat{}\ (length\ xs) - G$
by (*simp add*: *right-diff-distrib'*)
also have $... < G * G\ \hat{}\ (length\ xs)$
using ‹$x < G$› *less-imp-Suc-add* **by** *fastforce*
also have $... = G\ \hat{}\ (Suc\ (length\ xs))$
by *simp*
finally have $encode\ (x\ \#\ xs) < G\ \hat{}\ (length\ (x\ \#\ xs))$
by *simp*
then show *?case* **.**
qed

Decoding a number into an m-tuple of numbers is then a well-defined operation.

definition *decode* :: $nat \Rightarrow nat \Rightarrow nat\ list$ **where**
$decode\ m\ n \equiv THE\ xs.\ encode\ xs = n \wedge length\ xs = m \wedge (\forall x \in set\ xs.\ x < G)$

lemma *decode*:
assumes $n < G\ \hat{}\ m$
shows *encode-decode*: $encode\ (decode\ m\ n) = n$
and *length-decode*: $length\ (decode\ m\ n) = m$
and *decode-less-G*: $\forall x \in set\ (decode\ m\ n).\ x < G$
proof −
have $\exists xs.\ length\ xs = m \wedge (\forall x \in set\ xs.\ x < G) \wedge encode\ xs = n$
using *assms encode-surj* **by** *simp*
then have $*$: $\exists !xs.\ encode\ xs = n \wedge length\ xs = m \wedge (\forall x \in set\ xs.\ x < G)$
using *encode-inj* **by** *auto*
let $?xs = decode\ m\ n$
let $?P = \lambda xs.\ encode\ xs = n \wedge length\ xs = m \wedge (\forall x \in set\ xs.\ x < G)$
have $encode\ ?xs = n \wedge length\ ?xs = m \wedge (\forall x \in set\ ?xs.\ x < G)$
using $*$ *theI'*[*of ?P*] *decode-def* **by** *simp*
then show $encode\ (decode\ m\ n) = n$ **and** $length\ (decode\ m\ n) = m$ **and** $\forall x \in set\ (decode\ m\ n).\ x < G$
by *simp-all*
qed

lemma *decode-encode*: $\forall x \in set\ xs.\ x < G \Longrightarrow decode\ (length\ xs)\ (encode\ xs) = xs$
proof −
fix *xs* :: *nat list*
assume *a*: $\forall x \in set\ xs.\ x < G$
then have $encode\ xs < G\ \hat{}\ (length\ xs)$
using *encode-less* **by** *simp*
show $decode\ (length\ xs)\ (encode\ xs) = xs$

unfolding *decode-def*
proof (*rule the-equality*)
show $encode\ xs = encode\ xs \wedge length\ xs = length\ xs \wedge (\forall x{\in}set\ xs.\ x < G)$
using *a* **by** *simp*
show $\bigwedge ys.\ encode\ ys = encode\ xs \wedge length\ ys = length\ xs \wedge (\forall x{\in}set\ ys.\ x < G) \Longrightarrow ys = xs$
using *a encode-inj* **by** *simp*
qed
qed

The simulator will access and update components of the encoded symbol.

definition *encode-nth* :: $nat \Rightarrow nat \Rightarrow nat \Rightarrow nat$ **where**
$encode\text{-}nth\ m\ n\ j \equiv decode\ m\ n\ !\ j$

definition *encode-upd* :: $nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow nat$ **where**
$encode\text{-}upd\ m\ n\ j\ x \equiv encode\ ((decode\ m\ n)\ [j{:=}x])$

lemma *encode-nth-less*:
assumes $n < G\ \hat{}\ m$ **and** $j < m$
shows $encode\text{-}nth\ m\ n\ j < G$
using *assms encode-nth-def decode-less-G length-decode* **by** *simp*

For the symbols the simulator actually uses, we fix $m = 2k+2$ and reserve the symbols $\triangleright$ and $\Box$, effectively shifting the symbols by two. We call the symbols $\{2, \ldots, G^{2k+2} + 2\}$ "code symbols".

definition *enc* :: $symbol\ list \Rightarrow symbol$ **where**
$enc\ xs \equiv encode\ xs + 2$

definition *dec* :: $symbol \Rightarrow symbol\ list$ **where**
$dec\ n \equiv decode\ (2 * k + 2)\ (n - 2)$

lemma *dec*:
assumes $n > 1$ **and** $n < G\ \hat{}(2 * k + 2) + 2$
shows *enc-dec*: $enc\ (dec\ n) = n$
and *length-dec*: $length\ (dec\ n) = 2 * k + 2$
and *dec-less-G*: $\forall x{\in}set\ (dec\ n).\ x < G$
proof −
show $enc\ (dec\ n) = n$
using *enc-def dec-def encode-decode assms* **by** *simp*
show $length\ (dec\ n) = 2 * k + 2$
using *enc-def dec-def length-decode assms* **by** *simp*
show $\forall x{\in}set\ (dec\ n).\ x < G$
using *enc-def dec-def decode-less-G assms*
by (*metis Suc-leI less-diff-conv2 one-add-one plus-1-eq-Suc*)
qed

lemma *dec-enc*: $\forall x{\in}set\ xs.\ x < G \Longrightarrow length\ xs = 2 * k + 2 \Longrightarrow dec\ (enc\ xs) = xs$
unfolding *enc-def dec-def* **using** *decode-encode* **by** *fastforce*

definition *enc-nth* :: $nat \Rightarrow nat \Rightarrow nat$ **where**
$enc\text{-}nth\ n\ j \equiv dec\ n\ !\ j$

lemma *enc-nth*: $\forall x{\in}set\ xs.\ x < G \Longrightarrow length\ xs = 2 * k + 2 \Longrightarrow enc\text{-}nth\ (enc\ xs)\ j = xs\ !\ j$
unfolding *enc-nth-def* **by** (*simp add*: *dec-enc*)

lemma *enc-nth-dec*:
assumes $n > 1$ **and** $n < G\ \hat{}(2 * k + 2) + 2$
shows $enc\text{-}nth\ n\ j = (dec\ n)\ !\ j$
using *assms enc-nth dec* **by** *metis*

abbreviation *is-code* :: $nat \Rightarrow bool$ **where**
$is\text{-}code\ n \equiv n < G\ \hat{}(2 * k + 2) + 2 \wedge 1 < n$

lemma *enc-nth-less*:
assumes $is\text{-}code\ n$ **and** $j < 2 * k + 2$

shows *enc-nth n j* < *G*
using *assms enc-nth-def* **by** (*metis dec-less-G in-set-conv-nth length-dec*)

lemma *enc-less*: ∀ *x*∈*set xs*. *x* < *G* ⟹ *length xs* = *2* ∗ *k* + *2* ⟹ *enc xs* < *G* ^ (*2* ∗ *k* + *2*) + *2*
using *encode-less enc-def* **by** *fastforce*

definition *enc-upd* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
enc-upd n j x ≡ *enc* ((*dec n*) [*j*:=*x*])

lemma *enc-upd-is-code*:
assumes *is-code n* **and** *j* < *2* ∗ *k* + *2* **and** *x* < *G*
shows *is-code* (*enc-upd n j x*)
proof −
let *?ys* = (*dec n*) [*j*:=*x*]
have ∀ *h*∈*set* (*dec n*). *h* < *G*
using *assms*(*1*,*2*) *dec-less-G* **by** *auto*
then have ∀ *h*∈*set ?ys*. *h* < *G*
using *assms*(*3*)
by (*metis in-set-conv-nth length-list-update nth-list-update-eq nth-list-update-neq*)
moreover have *length ?ys* = *2* ∗ *k* + *2*
using *assms length-dec* **by** *simp*
ultimately have *enc ?ys* < *G* ^ (*2* ∗ *k* + *2*) + *2*
using *enc-less* **by** *simp*
then show *?thesis*
using *enc-upd-def enc-def* **by** *simp*
qed

The code symbols require the simulator to have an alphabet of at least size $G^{2k+2}+2$. On top of that we want to store on a memorization tape the state that M is in. So the alphabet must have at least *length M* + *1* symbols. Both conditions are met by the simulator having an alphabet of size G':

definition *G′* :: *nat* **where**
G′ ≡ *G* ^ (*2* ∗ *k* + *2*) + *2* + *length M*

lemma *G′-ge-6*: *G′* ≥ *6*
proof −
have *4* ^ *2* > (*5*::*nat*)
by *simp*
then have *G* ^ *2* > (*5*::*nat*)
using *G-ge-4 less-le-trans power2-nat-le-eq-le* **by** *blast*
then have *G* ^ (*2* ∗ *k* + *2*) > (*5*::*nat*)
using *k-ge-2 G-ge-4*
by (*metis* (*no-types*, *opaque-lifting*) *dec-induct le-add2 order-less-le-subst1 pow-mono zero-less-Suc zero-less-numeral*)
then show *?thesis*
using *G′-def* **by** *simp*
qed

corollary *G′-ge*: *G′* ≥ *4 G′* ≥ *5*
using *G′-ge-6* **by** *simp-all*

lemma *G′-ge-G*: *G′* ≥ *G*
proof −
have *G* ^ *2* ≥ *G*
by (*simp add*: *power2-nat-le-imp-le*)
then have *G* ^ (*2* ∗ *k* + *2*) ≥ *G*
by *simp*
then show *?thesis* **using** *G′-def*
by *linarith*
qed

corollary *enc-less-G′*: ∀ *x*∈*set xs*. *x* < *G* ⟹ *length xs* = *2* ∗ *k* + *2* ⟹ *enc xs* < *G′*
using *enc-less G′-def* **by** *fastforce*

lemma *enc-greater*: *enc xs* > *1*

using *enc-def* **by** *simp*

5.3.2 Construction of the simulator Turing machine

The simulator is a sequence of three Turing machines: The "formatter", which initializes the output tape; the loop, which simulates the TM M for polynomially many steps; and a cleanup TM, which makes the output tape look like the output tape of M. All these machines are discussed in order in the following subsections.

The simulator will start with $2k+1$ memorization tapes for a total of $2k+3$ tapes. The simulation action will take place on the output tape.

Initializing the simulator's tapes

The function T' is polynomially bounded and therefore there is a polynomial-time two-tape oblivious Turing machine that outputs at least $T'(n)$ symbols **1** on an input of length n, as we have proven in the previous section (lemma *polystructor*). We now obtain such a Turing machine together with its running time bound and trace. This TM will be one of our blocks for building the simulator TM. We will call it the "formatter".

definition *fmt-es-pM* :: (*nat* ⇒ *nat*) × (*nat* ⇒ (*nat* × *nat*) *list*) × *machine* **where**
fmt-es-pM ≡ *SOME tec.*
turing-machine 2 G′ (*snd* (*snd tec*)) ∧
big-oh-poly (*λn. length* ((*fst* (*snd tec*)) *n*)) ∧
big-oh-poly (*fst tec*) ∧
(∀ *n. T′ n* ≤ (*fst tec*) *n*) ∧
(∀ *zs. proper-symbols zs* ⟶ *traces* (*snd* (*snd tec*)) (*start-tapes-2 zs*) ((*fst* (*snd tec*)) (*length zs*)) (*one-tapes-2 zs* ((*fst tec*) (*length zs*))))

lemma *polystructor′*:
fixes *GG* :: *nat* **and** *g* :: *nat* ⇒ *nat*
assumes *big-oh-poly g* **and** *GG* ≥ *5*
shows ∃ *f-es-M*.
turing-machine 2 GG (*snd* (*snd f-es-M*)) ∧
big-oh-poly (*λn. length* ((*fst* (*snd f-es-M*)) *n*)) ∧
big-oh-poly (*fst f-es-M*) ∧
(∀ *n. g n* ≤ (*fst f-es-M*) *n*) ∧
(∀ *zs. proper-symbols zs* ⟶ *traces* (*snd* (*snd f-es-M*)) (*start-tapes-2 zs*) ((*fst* (*snd f-es-M*)) (*length zs*)) (*one-tapes-2 zs* ((*fst f-es-M*) (*length zs*))))
using *polystructor*[*OF assms*] **by** *auto*

lemma *fmt-es-pM*: *turing-machine 2 G′* (*snd* (*snd fmt-es-pM*)) ∧
big-oh-poly (*λn. length* ((*fst* (*snd fmt-es-pM*)) *n*)) ∧
big-oh-poly (*fst fmt-es-pM*) ∧
(∀ *n. T′ n* ≤ (*fst fmt-es-pM*) *n*) ∧
(∀ *zs. proper-symbols zs* ⟶ *traces* (*snd* (*snd fmt-es-pM*)) (*start-tapes-2 zs*) ((*fst* (*snd fmt-es-pM*)) (*length zs*)) (*one-tapes-2 zs* ((*fst fmt-es-pM*) (*length zs*))))
using *fmt-es-pM-def polystructor′*[*OF poly-T′ G′-ge*(*2*)]
someI-ex[*of λtec.*
turing-machine 2 G′ (*snd* (*snd tec*)) ∧
big-oh-poly (*λn. length* ((*fst* (*snd tec*)) *n*)) ∧
big-oh-poly (*fst tec*) ∧
(∀ *n. T′ n* ≤ (*fst tec*) *n*) ∧
(∀ *zs. proper-symbols zs* ⟶ *traces* (*snd* (*snd tec*)) (*start-tapes-2 zs*) ((*fst* (*snd tec*)) (*length zs*)) (*one-tapes-2 zs* ((*fst tec*) (*length zs*))))]
by *simp*

definition *fmt* :: *nat* ⇒ *nat* **where**
fmt ≡ *fst fmt-es-pM*

definition *es-fmt* :: *nat* ⇒ (*nat* × *nat*) *list* **where**
es-fmt ≡ *fst* (*snd fmt-es-pM*)

definition *tm-fmt* :: *machine* **where**
tm-fmt ≡ *snd* (*snd fmt-es-pM*)

The formatter TM is *tm-fmt*, the number of **1** symbols written to the output tape on input of length n is *fmt n*, and the trace on inputs of length n is *es-fmt n*.

lemma *fmt*:
turing-machine 2 G′ tm-fmt
big-oh-poly (λ*n. length* (*es-fmt n*))
big-oh-poly fmt
⋀*n. T′ n* ≤ *fmt n*
⋀*zs. proper-symbols zs* ⟹
traces tm-fmt (*start-tapes-2 zs*) (*es-fmt* (*length zs*)) (*one-tapes-2 zs* (*fmt* (*length zs*)))
unfolding *fmt-def es-fmt-def tm-fmt-def* **using** *fmt-es-pM* **by** *simp-all*

lemma *fmt-ge-n*: *fmt n* ≥ *n*
using *fmt*(*4*) *T′-def* **by** (*metis dual-order.strict-trans2 le-less-linear not-add-less2*)

The formatter is a two-tape TM. The first incarnation of the simulator will have two tapes and $2k + 1$ memorization tapes. So for now we formally need to extend the formatter to $2k + 3$ tapes:

definition *tm1* ≡ *append-tapes 2* (*2* ∗ *k* + *3*) *tm-fmt*

lemma *tm1-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tm1*
unfolding *tm1-def* **using** *append-tapes-tm* **by** (*simp add*: *fmt*(*1*))

Next we replace all non-blank symbols on the output tape by code symbols representing the tuple of $2k + 2$ zeros.

definition *tm1-2* ≡ *tm-const-until 1 1* {□} (*enc* (*replicate k 0* @ *replicate k 0* @ [*0*, *0*]))

lemma *tm1-2-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tm1-2*
unfolding *tm1-2-def*
using *G′-ge*
proof (*intro tm-const-until-tm*, *auto*)
show *enc* (*replicate k 0* @ *replicate k 0* @ [*0*, *0*]) < *G′*
using *G-ge-4* **by** (*intro enc-less-G′*, *auto*)
qed

definition *tm2* ≡ *tm1* ;; *tm1-2*

lemma *tm2-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tm2*
unfolding *tm2-def* **using** *tm1-tm tm1-2-tm* **by** *simp*

definition *tm3* ≡ *tm2* ;; *tm-start 1*

lemma *tm3-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tm3*
unfolding *tm3-def* **using** *tm2-tm tm-start-tm G′-ge* **by** *simp*

Back at the start symbol of the output tape, we replace it by the code symbol for the tuple 1^k1^k01. The first k ones represent that initially the k tapes of M have the start symbol (numerically 1) in the leftmost cell. The second run of k ones represent that initially all k tapes have their tape heads in the leftmost cell. The following 0 is the first bit of the unary counter, currently set to zero. The final flag 1 signals that this is the leftmost cell. Unlike the start symbols this signal flag cannot be overwritten by M.

definition *tm4* ≡ *tm3* ;; *tm-write 1* (*enc* (*replicate k 1* @ *replicate k 1* @ [*0*, *1*]))

lemma *tm3-4-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′* (*tm-write 1* (*enc* (*replicate k 1* @ *replicate k 1* @ [*0*, *1*])))
using *G′-ge enc-less-G′ G-ge-4 tm-write-tm* **by** *simp*

lemma *tm4-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tm4*
unfolding *tm4-def* **using** *tm3-tm tm3-4-tm* **by** *simp*

definition *tm5* ≡ *tm4* ;; *tm-right 1*

lemma *tm5-tm*: *turing-machine* $(2 * k + 3)$ G' *tm5*
unfolding *tm5-def* **using** *tm4-tm* **by** (*simp add*: G'*-ge*(*1*) *tm-right-tm*)

So far the simulator's output tape encodes k tapes that are empty but for the start symbols. To represent the start configuration of M, we need to copy the contents of the input tape to the output tape. The following TM moves the work head to the first blank while copying the input tape content. Here we exploit $T'(n) \geq n$, which implies that the formatted section is long enough to hold the input.

definition *tm5-6* $\equiv$ *tm-trans-until 0 1* {*0*} (λh. *enc* (*h mod G* # *replicate* ($k - 1$) *0* @ *replicate k 0* @ [*0*, *0*]))

definition *tm6* $\equiv$ *tm5* ;; *tm5-6*

lemma *tm5-6-tm*: *turing-machine* $(2 * k + 3)$ G' *tm5-6*
unfolding *tm5-6-def*
proof (*rule tm-trans-until-tm*, *auto simp add*: G'*-ge*(*1*) *G-ge-4 k-ge-2 enc-less-*G')
show $\bigwedge h.\ h < G' \Longrightarrow$ *enc* (*h mod G* # *replicate* (k − *Suc 0*) *0* @ *replicate k 0* @ [*0*, *0*]) $< G'$
using *G-ge-4 k-ge-2 enc-less-*G' **by** *simp*
qed

lemma *tm6-tm*: *turing-machine* $(2 * k + 3)$ G' *tm6*
unfolding *tm6-def* **using** *tm5-tm tm5-6-tm* **by** *simp*

Since we have overwritten the leftmost cell of the output tape with some code symbol, we cannot return to it using *tm-start*. But we can use *tm-left-until* with a special set of symbols:

abbreviation *StartSym* :: *symbol set* **where**
StartSym $\equiv \{y.\ y < G \,\hat{}\, (2 * k + 2) + 2 \wedge y > 1 \wedge$ *dec* y ! $(2 * k + 1) = 1\}$

abbreviation *tm-left-until1* $\equiv$ *tm-left-until StartSym 1*

lemma *tm-left-until1-tm*: *turing-machine* $(2 * k + 3)$ G' *tm-left-until1*
using *tm-left-until-tm* G'*-ge*(*1*) *k-ge-2* **by** *simp*

definition *tm7* $\equiv$ *tm6* ;; *tm-left-until1*

lemma *tm7-tm*: *turing-machine* $(2 * k + 3)$ G' *tm7*
unfolding *tm7-def* **using** *tm6-tm tm-left-until1-tm* **by** *simp*

Tape number 2 is meant to memorize M's state during the simulation. Initially the state is the start state, that is, zero.

definition *tm8* $\equiv$ *tm7* ;; *tm-write 2 0*

lemma *tm8-tm*: *turing-machine* $(2 * k + 3)$ G' *tm8*
unfolding *tm8-def* **using** *tm7-tm tm-write-tm k-ge-2* G'*-ge*(*1*) **by** *simp*

We also initialize memorization tapes $3, \ldots, 2k + 2$ to zero. This concludes the initialization of the simulator's tapes.

definition *tm9* $\equiv$ *tm8* ;; *tm-write-many* {*3*..<$2 * k + 3$} *0*

lemma *tm9-tm*: *turing-machine* $(2 * k + 3)$ G' *tm9*
unfolding *tm9-def* **using** *tm8-tm tm-write-many-tm k-ge-2* G'*-ge*(*1*) **by** *simp*

The loop

The core of the simulator is a loop whose body is executed *fmt n* many times. Each iteration simulates one step of the Turing machine M. For the analysis of the loop we describe the $2k + 3$ tapes in the form [*a*, *b*, *c*] @ *map f1* [*0*..<*k*] @ *map f2* [*0*..<*k*].

lemma *threeplus2k-2*:
assumes $3 \leq j \wedge j < k + 3$
shows ([*a*, *b*, *c*] @ *map f1* [*0*..<*k*] @ *map f2* [*0*..<*k*]) ! j = *f1* ($j - 3$)
using *assms* **by** (*simp add*: *nth-append less-diff-conv2 numeral-3-eq-3*)

lemma *threeplus2k-3*:

assumes $k + 3 \leq j \wedge j < 2 * k + 3$
shows $([a, b, c] @ map\ f1\ [0..<k] @ map\ f2\ [0..<k])\ !\ j = f2\ (j - k - 3)$
using *assms* **by** (*simp add*: *nth-append less-diff-conv2 numeral-3-eq-3*)

To ensure the loop runs for *fmt n* iterations we increment the unary counter in the code symbols in each iteration. The loop terminates when there are no more code symbols with an unset counter flag. The TM that prepares the loop condition sweeps the output tape left-to-right searching for the first symbol that is either blank or has an unset counter flag. The condition then checks for which of the two cases occurred. This is the condition TM:

definition $tmC \equiv tm\text{-}right\text{-}until\ 1\ \{y.\ y < G\ \hat{}\ (2 * k + 2) + 2 \wedge (y = 0 \vee dec\ y\ !\ (2 * k) = 0)\}$

lemma *tmC-tm*: $turing\text{-}machine\ (2 * k + 3)\ G'\ tmC$
using *tmC-def tm-right-until-tm* **using** $G'\text{-}ge(1)$ **by** *simp*

At the start of the iteration, the memorization tape 2 has the state of M, and all other memorization tapes contain 0. The output tape head is at the leftmost code symbol with unset counter flag. The first order of business is to move back to the beginning of the output tape.

definition $tmL1 \equiv tm\text{-}left\text{-}until1$

lemma *tmL1-tm*: $turing\text{-}machine\ (2 * k + 3)\ G'\ tmL1$
unfolding *tmL1-def* **using** *tm6-tm tm-left-until1-tm* **by** *simp*

Then the output tape head sweeps right until it encounters a blank. During the sweep the Turing machine checks for any set head flags, and if it finds the one for tape j set, it memorizes the symbol for tape j on tape $3 + k + j$. The next command performs this operation:

definition *cmdL2* :: *command* **where**
cmdL2 rs ≡
(*if rs* ! *1* = □
then (*1*, *zip rs* (*replicate* (*2*∗*k*+*3*) *Stay*))
else
(*0*,
[(*rs*!*0*, *Stay*), (*rs*!*1*, *Right*), (*rs*!*2*, *Stay*)] @
(*map* (λ*j*. (*rs* ! (*j* + *3*), *Stay*)) [*0*..<*k*]) @
(*map* (λ*j*. (*if 1* < *rs* ! *1* ∧ *rs* ! *1* < *G*^(*2*∗*k*+*2*)+*2* ∧ *enc-nth* (*rs*!*1*) (*k*+*j*) = *1 then enc-nth* (*rs*!*1*) *j else rs*!(*3*+*k*+*j*), *Stay*)) [*0*..<*k*])))

lemma *cmdL2-at-eq-0*:
assumes $rs\ !\ 1 = 0$ **and** $j < 2 * k + 3$ **and** $length\ rs = 2 * k + 3$
shows $snd\ (cmdL2\ rs)\ !\ j = (rs\ !\ j, Stay)$
using *assms* **by** (*simp add*: *cmdL2-def*)

lemma *cmdL2-at-3*:
assumes $rs\ !\ 1 \neq \square$ **and** $3 \leq j \wedge j < k + 3$
shows $cmdL2\ rs\ [!]\ j = (rs\ !\ j, Stay)$
using *cmdL2-def assms threeplus2k-2*
by (*metis* (*no-types*, *lifting*) *le-add-diff-inverse2 snd-conv*)

lemma *cmdL2-at-4*:
assumes $rs\ !\ 1 \neq \square$ **and** $k + 3 \leq j \wedge j < 2 * k + 3$
shows $cmdL2\ rs\ [!]\ j =$
(*if 1* < *rs* ! *1* ∧ *rs* ! *1* < *G*^(*2*∗*k*+*2*)+*2* ∧ *enc-nth* (*rs* ! *1*) (*j*−*3*) = *1*
then enc-nth (*rs* ! *1*) (*j*−*k*−*3*)
else rs ! *j*, *Stay*)
unfolding *cmdL2-def* **using** *assms threeplus2k-3*[*OF assms*(*2*), *of* (*rs* ! *0*, *Stay*)] **by** *simp*

lemma *cmdL2-at-4''*:
assumes $rs\ !\ 1 \neq \square$
and $k + 3 \leq j \wedge j < 2 * k + 3$
and ¬ (*1* < *rs* ! *1* ∧ *rs* ! *1* < *G*^(*2*∗*k*+*2*)+*2*)
shows $cmdL2\ rs\ [!]\ j = (rs\ !\ j, Stay)$
proof −
have $cmdL2\ rs\ [!]\ j =$

(*if 1 < rs ! 1 ∧ rs ! 1 < G^(2∗k+2)+2 ∧ enc-nth (rs!1) (j−3) = 1 then enc-nth (rs!1) (j−k−3) else rs!j*, *Stay*)
    using *assms cmdL2-at-4* **by** *blast*
  then show *?thesis*
    using *assms*(*3*) **by** *auto*
qed

lemma *turing-command-cmdL2*: *turing-command* (*2 ∗ k + 3*) *1 G′ cmdL2*
proof
  show ⋀*gs. length gs = 2 ∗ k + 3 ⟹ length* ([!!] *cmdL2 gs*) = *length gs*
    unfolding *cmdL2-def* **by** *simp*
  show ⋀*gs. length gs = 2 ∗ k + 3 ⟹ 0 < 2 ∗ k + 3 ⟹ cmdL2 gs* [.] *0 = gs ! 0*
    unfolding *cmdL2-def* **by** *simp*
  show *cmdL2 gs* [.] *j < G′*
      if *length gs = 2 ∗ k + 3* ⋀*i. i < length gs ⟹ gs ! i < G′ j < length gs*
      for *gs j*
  proof (*cases gs ! 1 = 0*)
    case *True*
    then have *cmdL2 gs* = (*1*, *zip gs* (*replicate* (*2∗k+3*) *Stay*))
      unfolding *cmdL2-def* **by** *simp*
    have *cmdL2 gs* [.] *j = gs ! j*
      using *that*(*1*,*3*) **by** (*simp add*: ‹*cmdL2 gs* = (*1*, *zip gs* (*replicate* (*2 ∗ k + 3*) *Stay*))›)
    then show *?thesis*
      using *that*(*2*,*3*) **by** *simp*
  next
    case *False*
    consider *j = 0* | *j = 1* | *j = 2* | *3 ≤ j ∧ j < k + 3* | *k + 3 ≤ j ∧ j < 2 ∗ k + 3*
      using ‹*j < length gs*› ‹*length gs = 2 ∗ k + 3*› **by** *linarith*
    then show *?thesis*
    proof (*cases*)
      case *1*
      then show *?thesis*
        by (*simp add*: *cmdL2-def that*(*1*) *that*(*2*))
    next
      case *2*
      then show *?thesis*
        unfolding *cmdL2-def* **using** *False that* **by** *auto*
    next
      case *3*
      then show *?thesis*
        unfolding *cmdL2-def* **using** *False that* **by** *auto*
    next
      case *4*
      then have *snd* (*cmdL2 gs*) ! *j* = (*gs ! j*, *Stay*)
        unfolding *cmdL2-def* **using** *False that threeplus2k-2*[*OF 4*, *of* (*gs ! 0*, *Stay*)] **by** *simp*
      then show *?thesis*
        using *that* **by** (*simp add*: ‹*snd* (*cmdL2 gs*) ! *j* = (*gs ! j*, *Stay*)›)
    next
      case *5*
      show *?thesis*
      proof (*cases 1 < gs ! 1 ∧ gs ! 1 < G ^*(*2∗k+2*) + *2*)
        case *True*
        then have ∗: *cmdL2 gs* [.] *j* = (*if enc-nth* (*gs ! 1*) (*j−3*) = *1 then enc-nth* (*gs ! 1*) (*j−k−3*) *else gs ! j*)
          using *5 False* **by** (*simp add*: *cmdL2-at-4*)
        let *?n = gs ! 1*
        have *len*: *length* (*dec ?n*) = *2 ∗ k + 2* **and** *less-G*: ∀*x*∈*set* (*dec ?n*). *x < G*
          using *True length-dec dec-less-G* **by** (*simp*, *blast*)
        have ∗∗: *enc-nth ?n* (*j−k−3*) = *dec ?n* ! (*j−k−3*)
          using *enc-nth-dec True* **by** *simp*
        then have *dec ?n* ! (*j−k−3*) < *G*
          using *less-G 5 len* **by** *auto*
        then have *dec ?n* ! (*j−k−3*) < *G′*
          using *G′-ge-G* **by** *simp*

moreover have *gs* ! *j* < *G′*
using *that* **by** *simp*
ultimately show *?thesis*
using ∗ ∗∗ **by** *simp*
next
case *6*: *False*
have *cmdL2 gs* [!] *j* = (*gs* ! *j*, *Stay*)
using *cmdL2-at-4′′*[*OF False 5 6*] **by** *simp*
then show *?thesis*
using *that* **by** (*simp add*: ‹*snd* (*cmdL2 gs*) ! *j* = (*gs* ! *j*, *Stay*)›)
qed
qed
qed
show ⋀*gs*. *length gs* = *2* ∗ *k* + *3* ⟹ [∗] (*cmdL2 gs*) ≤ *1*
using *cmdL2-def* **by** *simp*
qed

definition *tmL1-2* ≡ [*cmdL2*]

lemma *tmL1-2-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tmL1-2*
using *tmL1-2-def* **using** *turing-command-cmdL2 G′-ge* **by** *auto*

definition *tmL2* ≡ *tmL1* ;; *tmL1-2*

lemma *tmL2-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tmL2*
by (*simp add*: *tmL1-2-tm tmL1-tm tmL2-def*)

The memorization tapes $3, \ldots, 2+k$ will store the head movements for tapes $0, \ldots, k-1$ of M. The directions are encoded as symbols thus:

definition *direction-to-symbol* :: *direction* ⇒ *symbol* **where**
direction-to-symbol m ≡ (*case m of Left* ⇒ □ | *Stay* ⇒ ▷ | *Right* ⇒ **0**)

lemma *direction-to-symbol-less*: *direction-to-symbol m* < *3*
using *direction-to-symbol-def* **by** (*cases m*) *simp-all*

At this point in the iteration the memorization tapes $k+3, \ldots, 2k+2$ contain the symbols under the k tape heads of M, and tape 2 contains the state M is in. Therefore all information is available to determine the actions M is taking in the step currently simulated. The next command has the entire behavior of M "hard-coded" and causes the actions to be stored on memorization tapes $3, \ldots, 2k+2$: the output symbols on the tapes $k+3, \ldots, 2k+2$, and the k head movements on the tapes $3, \ldots, k+2$. The follow-up state will again be memorized on tape 2. All this happens in one step of the simulator machine.

definition *cmdL3* :: *command* **where**
cmdL3 rs ≡
(*1*,
[(*rs* ! *0*, *Stay*),
(*rs* ! *1*, *Stay*),
(*if rs* ! *2* < *length M* ∧ (∀ *h*∈*set* (*drop* (*k* + *3*) *rs*). *h* < *G*)
then fst ((*M* ! (*rs* ! *2*)) (*drop* (*k* + *3*) *rs*))
else rs ! *2*, *Stay*)] @
map
(λ*j*. (*if rs* ! *2* < *length M* ∧ (∀ *h*∈*set* (*drop* (*k* + *3*) *rs*). *h* < *G*) *then direction-to-symbol* ((*M* ! (*rs* ! *2*)) (*drop* (*k* + *3*) *rs*) [~] *j*) *else 1*, *Stay*))
[*0*..<*k*] @
map (λ*j*. (*if rs* ! *2* < *length M* ∧ (∀ *h*∈*set* (*drop* (*k* + *3*) *rs*). *h* < *G*) *then* ((*M* ! (*rs* ! *2*)) (*drop* (*k* + *3*) *rs*) [.] *j*) *else rs* ! (*k* + *3* + *j*), *Stay*)) [*0*..<*k*])

lemma *cmdL3-at-2a*:
assumes *gs* ! *2* < *length M* ∧ (∀ *h*∈*set* (*drop* (*k* + *3*) *gs*). *h* < *G*)
shows *cmdL3 gs* [!] *2* = (*fst* ((*M* ! (*gs* ! *2*)) (*drop* (*k* + *3*) *gs*)), *Stay*)
using *cmdL3-def assms* **by** *simp*

lemma *cmdL3-at-2b*:

assumes $\neg$ (*gs* ! *2* < *length M* $\wedge$ ($\forall$ *h*$\in$*set* (*drop* (*k* + *3*) *gs*). *h* < *G*))
shows *cmdL3 gs* [!] *2* = (*gs* ! *2*, *Stay*)
using *cmdL3-def assms* **by** *auto*

lemma *cmdL3-at-3a*:
assumes *3* $\leq$ *j* $\wedge$ *j* < *k* + *3*
and *gs* ! *2* < *length M* $\wedge$ ($\forall$ *h*$\in$*set* (*drop* (*k* + *3*) *gs*). *h* < *G*)
shows *cmdL3 gs* [!] *j* = (*direction-to-symbol* ((*M* ! (*gs* ! *2*)) (*drop* (*k* + *3*) *gs*) [~] (*j* − *3*)), *Stay*)
using *cmdL3-def assms*(*2*) *threeplus2k-2*[*OF assms*(*1*), *of* (*gs* ! *0*, *Stay*)] **by** *simp*

lemma *cmdL3-at-3b*:
assumes *3* $\leq$ *j* $\wedge$ *j* < *k* + *3*
and $\neg$ (*gs* ! *2* < *length M* $\wedge$ ($\forall$ *h*$\in$*set* (*drop* (*k* + *3*) *gs*). *h* < *G*))
shows *cmdL3 gs* [!] *j* = (*1*, *Stay*)
using *cmdL3-def assms*(*2*) *threeplus2k-2*[*OF assms*(*1*), *of* (*gs* ! *0*, *Stay*)] **by** *auto*

lemma *cmdL3-at-4a*:
assumes *k* + *3* $\leq$ *j* $\wedge$ *j* < *2* ∗ *k* + *3*
and *gs* ! *2* < *length M* $\wedge$ ($\forall$ *h*$\in$*set* (*drop* (*k* + *3*) *gs*). *h* < *G*)
shows *cmdL3 gs* [!] *j* = ((*M* ! (*gs* ! *2*)) (*drop* (*k* + *3*) *gs*) [.] (*j* − *k* − *3*), *Stay*)
using *cmdL3-def assms*(*2*) *threeplus2k-3*[*OF assms*(*1*), *of* (*gs* ! *0*, *Stay*)] **by** *simp*

lemma *cmdL3-at-4b*:
assumes *k* + *3* $\leq$ *j* $\wedge$ *j* < *2* ∗ *k* + *3*
and $\neg$ (*gs* ! *2* < *length M* $\wedge$ ($\forall$ *h*$\in$*set* (*drop* (*k* + *3*) *gs*). *h* < *G*))
shows *cmdL3 gs* [!] *j* = (*gs* ! *j*, *Stay*)
using *assms cmdL3-def threeplus2k-3*[*OF assms*(*1*), *of* (*gs* ! *0*, *Stay*)] **by** *auto*

lemma *cmdL3-if-comm*:
assumes *length gs* = *2* ∗ *k* + *3* **and** *gs* ! *2* < *length M* $\wedge$ ($\forall$ *h*$\in$*set* (*drop* (*k* + *3*) *gs*). *h* < *G*)
shows *length* ([!!] (*M* ! (*gs* ! *2*)) (*drop* (*k* + *3*) *gs*)) = *k*
and $\bigwedge$*j*. *j* < *k* $\Longrightarrow$ (*M* ! (*gs* ! *2*)) (*drop* (*k* + *3*) *gs*) [.] *j* < *G*
proof −
let *?rs* = *drop* (*k* + *3*) *gs*
let *?cmd* = *M* ! (*gs* ! *2*)
have ∗: *turing-command k* (*length M*) *G ?cmd*
using *assms*(*2*) *tm-M turing-machineD*(*3*) **by** *simp*
then show *length* ([!!] *?cmd ?rs*) = *k*
using *turing-commandD*(*1*) *assms*(*1*) **by** *simp*
have $\bigwedge$*i*. *i* < *length ?rs* $\Longrightarrow$ *?rs* ! *i* < *G*
using *assms*(*2*) *nth-mem* **by** *blast*
then show $\bigwedge$*j*. *j* < *k* $\Longrightarrow$ (*M* ! (*gs* ! *2*)) (*drop* (*k* + *3*) *gs*) [.] *j* < *G*
using ∗ *turing-commandD*(*2*) *assms* **by** *simp*
qed

lemma *turing-command-cmdL3*: *turing-command* (*2* ∗ *k* + *3*) *1 G′ cmdL3*
proof
show $\bigwedge$*gs*. *length gs* = *2* ∗ *k* + *3* $\Longrightarrow$ *length* ([!!] *cmdL3 gs*) = *length gs*
using *cmdL3-def* **by** *simp*
show $\bigwedge$*gs*. *length gs* = *2* ∗ *k* + *3* $\Longrightarrow$ *0* < *2* ∗ *k* + *3* $\Longrightarrow$ *cmdL3 gs* [.] *0* = *gs* ! *0*
using *cmdL3-def* **by** *simp*
show *cmdL3 gs* [.] *j* < *G′*
if *length gs* = *2* ∗ *k* + *3* $\bigwedge$*i*. *i* < *length gs* $\Longrightarrow$ *gs* ! *i* < *G′ j* < *length gs*
for *gs j*
proof −
consider *j* = *0* | *j* = *1* | *j* = *2* | *3* $\leq$ *j* $\wedge$ *j* < *k* + *3* | *k* + *3* $\leq$ *j* $\wedge$ *j* < *2* ∗ *k* + *3*
using ‹*j* < *length gs*› ‹*length gs* = *2* ∗ *k* + *3*› **by** *linarith*
then show *?thesis*
proof (*cases*)
case *1*
then show *?thesis*
using *that cmdL3-def* **by** *simp*
next

case *2*
then show *?thesis*
using *that cmdL3-def* **by** *simp*
next
case *3*
then show *?thesis*
proof (*cases gs* ! *2* < *length M* ∧ (∀ *h*∈*set* (*drop* (*k* + *3*) *gs*). *h* < *G*))
case *True*
have *length* (*drop* (*k* + *3*) *gs*) = *k*
using *that*(*1*) **by** *simp*
then have *fst* ((*M* ! (*gs* ! *2*)) (*drop* (*k* + *3*) *gs*)) ≤ *length M*
using *True turing-commandD*(*4*) *tm-M turing-machineD*(*3*) **by** *blast*
moreover have *cmdL3 gs* [.] *j* = *fst* ((*M* ! (*gs* ! *2*)) (*drop* (*k* + *3*) *gs*))
using *cmdL3-def True 3* **by** *simp*
ultimately show *?thesis*
using *G′-def* **by** *simp*
next
case *False*
then have *cmdL3 gs* [.] *j* = *gs* ! *2*
using *cmdL3-def 3* **by** *auto*
then show *?thesis*
using *3 that*(*2*,*3*) **by** *simp*
qed
next
case *4*
then show *?thesis*
proof (*cases gs* ! *2* < *length M* ∧ (∀ *h*∈*set* (*drop* (*k* + *3*) *gs*). *h* < *G*))
case *True*
then have *cmdL3 gs* [.] *j* = *direction-to-symbol* ((*M* ! (*gs* ! *2*)) (*drop* (*k* + *3*) *gs*) [~] (*j* − *3*))
using *cmdL3-at-3a 4* **by** *simp*
then have *cmdL3 gs* [.] *j* < *3*
using *direction-to-symbol-less* **by** *simp*
then show *?thesis*
using *G′-ge* **by** *simp*
next
case *False*
then show *?thesis*
using *cmdL3-at-3b*[*OF 4*] *G′-ge* **by** *simp*
qed
next
case *5*
then show *?thesis*
proof (*cases gs* ! *2* < *length M* ∧ (∀ *h*∈*set* (*drop* (*k* + *3*) *gs*). *h* < *G*))
case *True*
then have *cmdL3 gs* [.] *j* = (*M* ! (*gs* ! *2*)) (*drop* (*k* + *3*) *gs*) [.] (*j* − *k* − *3*)
using *cmdL3-at-4a*[*OF 5*] **by** *simp*
then have *cmdL3 gs* [.] *j* < *G*
using *cmdL3-if-comm True 5 that*(*1*) **by** *auto*
then show *?thesis*
using *G′-ge-G* **by** *simp*
next
case *False*
then have *cmdL3 gs* [.] *j* = *gs* ! *j*
using *cmdL3-at-4b*[*OF 5*] **by** *simp*
then show *?thesis*
using *that* **by** *simp*
qed
qed
qed
show ⋀*gs*. *length gs* = *2* ∗ *k* + *3* ⟹ [∗] (*cmdL3 gs*) ≤ *1*
using *cmdL3-def* **by** *simp*
qed

definition *tmL2-3* ≡ [*cmdL3*]

lemma *tmL2-3-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′* *tmL2-3*
 unfolding *tmL2-3-def* **using** *G′-ge*(*1*) *turing-command-cmdL3* **by** *auto*

definition *tmL3* ≡ *tmL2* ;; *tmL2-3*

lemma *tmL3-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′* *tmL3*
 by (*simp add*: *tmL2-3-tm tmL2-tm tmL3-def*)

Next the output tape head sweeps left to the beginning of the tape, otherwise doing nothing.

definition *tmL4* ≡ *tmL3* ;; *tm-left-until1*

lemma *tmL4-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′* *tmL4*
 using *tmL3-tm tmL4-def tmL1-def tm-left-until1-tm* **by** *simp*

The next four commands *cmdL4*, *cmdL5*, *cmdL6*, *cmdL7* are parameterized by $jj = 0, \ldots, k - 1$. Their job is applying the write operation and head movement for tape jj of M. The entire block of commands will then be executed k times, once for each jj.

The first of these commands sweeps right again and applies the write operation for tape jj, which is memorized on tape $3 + k + jj$. To this end it checks for head flags and updates the code symbol component jj with the contents of tape $3 + k + jj$ when the head flag for tape jj is set.

definition *cmdL5 jj rs* ≡
```
  if rs ! 1 = □
  then (1, zip rs (replicate (2∗k+3) Stay))
  else
   (0,
    [(rs ! 0, Stay),
     (if is-code (rs ! 1) ∧ rs ! (3+k+jj) < G ∧ enc-nth (rs ! 1) (k+jj) = 1
      then enc-upd (rs ! 1) jj (rs ! (3+k+jj))
      else rs ! 1,
      Right),
     (rs ! 2, Stay)] @
     (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
     (map (λj. (rs ! (3 + k + j), Stay)) [0..<k]))
```

lemma *cmdL5-eq-0*:
 assumes *j* < *2* ∗ *k* + *3* **and** *length gs* = *2* ∗ *k* + *3* **and** *gs* ! *1* = *0*
 shows *cmdL5 jj gs* [!] *j* = (*gs* ! *j*, *Stay*)
 unfolding *cmdL5-def* **using** *assms* **by** *simp*

lemma *cmdL5-at-0*:
 assumes *gs* ! *1* ≠ *0*
 shows *cmdL5 jj gs* [!] *0* = (*gs* ! *0*, *Stay*)
 unfolding *cmdL5-def* **using** *assms* **by** *simp*

lemma *cmdL5-at-1*:
 assumes *gs* ! *1* ≠ *0*
  and *is-code* (*gs* ! *1*) ∧ *gs* ! (*3*+*k*+*jj*) < *G* ∧ *enc-nth* (*gs*!*1*) (*k*+*jj*) = *1*
 shows *cmdL5 jj gs* [!] *1* = (*enc-upd* (*gs*!*1*) *jj* (*gs*!(*3*+*k*+*jj*)), *Right*)
 unfolding *cmdL5-def* **using** *assms* **by** *simp*

lemma *cmdL5-at-1-else*:
 assumes *gs* ! *1* ≠ *0*
  and ¬ (*is-code* (*gs* ! *1*) ∧ *gs* ! (*3*+*k*+*jj*) < *G* ∧ *enc-nth* (*gs*!*1*) (*k*+*jj*) = *1*)
 shows *cmdL5 jj gs* [!] *1* = (*gs* ! *1*, *Right*)
 unfolding *cmdL5-def* **using** *assms* **by** *auto*

lemma *cmdL5-at-2*:
 assumes *gs* ! *1* ≠ *0*
 shows *cmdL5 jj gs* [!] *2* = (*gs* ! *2*, *Stay*)
 unfolding *cmdL5-def* **using** *assms* **by** *simp*

lemma *cmdL5-at-3*:
  assumes $gs\ !\ 1 \neq 0$ **and** $3 \leq j \wedge j < 3 + k$
  shows *cmdL5 jj gs* [!] $j = (gs\ !\ j,\ Stay)$
  unfolding *cmdL5-def* **using** *assms threeplus2k-2*[**where** $?a{=}(gs\ !\ 0,\ Stay)$] **by** *simp*

lemma *cmdL5-at-4*:
  assumes $gs\ !\ 1 \neq 0$ **and** $3 + k \leq j \wedge j < 2 * k + 3$
  shows *cmdL5 jj gs* [!] $j = (gs\ !\ j,\ Stay)$
  unfolding *cmdL5-def* **using** *assms threeplus2k-3*[**where** $?a{=}(gs\ !\ 0,\ Stay)$] **by** *simp*

lemma *turing-command-cmdL5*:
  assumes $jj < k$
  shows *turing-command* $(2 * k + 3)\ 1\ G'\ (cmdL5\ jj)$
proof
  show $length\ gs = 2 * k + 3 \Longrightarrow length\ ([!!]\ cmdL5\ jj\ gs) = length\ gs$ **for** *gs*
    unfolding *cmdL5-def* **by** $(cases\ gs!1{=}0)$ *simp-all*
  show *goal2*: $length\ gs = 2 * k + 3 \Longrightarrow 0 < 2 * k + 3 \Longrightarrow cmdL5\ jj\ gs\ [.]\ 0 = gs\ !\ 0$ **for** *gs*
    unfolding *cmdL5-def* **by** $(cases\ gs\ !\ 1{=}0)$ *simp-all*
  show *cmdL5 jj gs* [.] $j < G'$
    if $length\ gs = 2 * k + 3\ \bigwedge i.\ i < length\ gs \Longrightarrow gs\ !\ i < G'\ j < length\ gs$
    for *gs j*
  proof $(cases\ gs\ !\ 1 = 0)$
    case *True*
    then show *?thesis*
      using *that cmdL5-eq-0* **by** *simp*
  next
    case *False*
    consider $j = 0 \mid j = 1 \mid j = 2 \mid 3 \leq j \wedge j < k + 3 \mid k + 3 \leq j \wedge j < 2 * k + 3$
      using ‹$length\ gs = 2 * k + 3$› ‹$j < length\ gs$› **by** *linarith*
    then show *?thesis*
    proof (*cases*)
      case *1*
      then show *?thesis*
        using *that goal2* **by** *simp*
    next
      case *2*
      show *?thesis*
      proof $(cases\ 1 < gs\ !\ 1 \wedge gs\ !\ 1 < G\hat{}(2{*}k{+}2){+}2 \wedge gs\ !\ (3{+}k{+}jj) < G \wedge enc\text{-}nth\ (gs\ !\ 1)\ (k{+}jj) = 1)$
        case *True*
        then have $*$: *cmdL5 jj gs* [.] $j = enc\text{-}upd\ (gs\ !\ 1)\ jj\ (gs\ !\ (3{+}k{+}jj))$
          using *2 cmdL5-at-1*[*OF False*] **by** *simp*
        let $?n = gs\ !\ 1$
        let $?xs = dec\ ?n$
        let $?ys = (dec\ ?n)\ [jj{:=}gs!(3{+}k{+}jj)]$
        have $gs\ !\ (3{+}k{+}jj) < G$
          using *True* **by** *simp*
        moreover have $\forall h{\in}set\ (dec\ ?n).\ h < G$
          using *True dec-less-G* **by** *auto*
        ultimately have $\forall h{\in}set\ ?ys.\ h < G$
          by (*metis in-set-conv-nth length-list-update nth-list-update-eq nth-list-update-neq*)
        moreover have $length\ ?ys = 2 * k + 2$
          using *True length-dec* **by** *simp*
        ultimately have $enc\ ?ys < G\ \hat{}\ (2 * k + 2) + 2$
          using *enc-less* **by** *simp*
        then show *?thesis*
          using $*$ **by** (*simp add*: *enc-upd-def* G'*-def*)
      next
        case *b*: *False*
        then show *?thesis*
          using *that cmdL5-at-1-else*[*OF False*] *2* **by** *simp*
      qed
    next
      case *3*

```
      then show ?thesis
        using that cmdL5-at-2[OF False] by simp
    next
      case 4
      then show ?thesis
        using that cmdL5-at-3[OF False] by simp
    next
      case 5
      then show ?thesis
        using that cmdL5-at-4[OF False] by simp
    qed
  qed
  show ⋀gs. length gs = 2 * k + 3 ⟹ [*] (cmdL5 jj gs) ≤ 1
   using cmdL5-def by (metis (no-types, lifting) One-nat-def fst-conv le-eq-less-or-eq plus-1-eq-Suc trans-le-add2)

qed

definition tmL45 :: nat ⇒ machine where
  tmL45 jj ≡ [cmdL5 jj]

lemma tmL45-tm:
  assumes jj < k
  shows turing-machine (2 * k + 3) G' (tmL45 jj)
  using assms G'-ge turing-command-cmdL5 tmL45-def by auto
```

We move the output tape head one cell to the left.

```
definition tmL46 jj ≡ tmL45 jj ;; tm-left 1

lemma tmL46-tm:
  assumes jj < k
  shows turing-machine (2 * k + 3) G' (tmL46 jj)
  using assms G'-ge tm-left-tm tmL45-tm tmL46-def tmL45-def by simp
```

The next command sweeps left and applies the head movement for tape jj if this is a movement to the left. To this end it checks for a set head flag in component $k + jj$ of the code symbol and clears it. It also memorizes that it just cleared the head flag by changing the symbol on memorization tape $3 + jj$ to the number 3, which is not used to encode any actual head movement. In the next step of the sweep it will recognize this 3 and set the head flag in component $k + jj$ of the code symbol. The net result is that the head flag for tape jj has moved one cell to the left.

```
abbreviation is-beginning :: symbol ⇒ bool where
  is-beginning y ≡ is-code y ∧ dec y ! (2 * k + 1) = 1

definition cmdL7 :: nat ⇒ command where
  cmdL7 jj rs ≡
 (if is-beginning (rs ! 1) then 1 else 0,
  if rs ! (3 + jj) = □ ∧ enc-nth (rs ! 1) (k + jj) = 1 ∧ is-code (rs ! 1) ∧ ¬ is-beginning (rs ! 1) then
   [(rs ! 0, Stay),
    (enc-upd (rs ! 1) (k + jj) 0, Left),
    (rs ! 2, Stay)] @
    (map (λj. (if j = jj then 3 else rs ! (j + 3), Stay)) [0..<k]) @
    (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
  else if rs ! (3 + jj) = 3 ∧ is-code (rs ! 1) then
   [(rs ! 0, Stay),
    (enc-upd (rs ! 1) (k + jj) 1, Left),
    (rs ! 2, Stay)] @
    (map (λj. (if j = jj then 0 else rs ! (j + 3), Stay)) [0..<k]) @
    (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
  else
   [(rs ! 0, Stay),
    (rs ! 1, Left),
    (rs ! 2, Stay)] @
    (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
    (map (λj. (rs ! (3 + k + j), Stay)) [0..<k]))
```

```
abbreviation condition7a gs jj ≡
  gs ! (3 + jj) = 0 ∧ enc-nth (gs ! 1) (k + jj) = 1 ∧ is-code (gs ! 1) ∧ ¬ is-beginning (gs ! 1)
abbreviation condition7b gs jj ≡
  ¬ condition7a gs jj ∧ gs ! (3 + jj) = 3 ∧ is-code (gs ! 1)
abbreviation condition7c gs jj ≡
  ¬ condition7a gs jj ∧ ¬ condition7b gs jj

lemma turing-command-cmdL7:
  assumes jj < k
  shows turing-command (2 ∗ k + 3) 1 G′ (cmdL7 jj)
proof
  show length ([!!] cmdL7 jj gs) = length gs if length gs = 2 ∗ k + 3 for gs
  proof −
    consider condition7a gs jj | condition7b gs jj | condition7c gs jj
      by blast
    then show ?thesis
      unfolding cmdL7-def using that by (cases) simp-all
  qed
  show goal2: 0 < 2 ∗ k + 3 =⇒ cmdL7 jj gs [.] 0 = gs ! 0 if length gs = 2 ∗ k + 3 for gs
  proof −
    consider condition7a gs jj | condition7b gs jj | condition7c gs jj
      by blast
    then show ?thesis
      unfolding cmdL7-def using that by (cases) simp-all
  qed
  show cmdL7 jj gs [.] j < G′
    if gs: j < length gs length gs = 2 ∗ k + 3 ⋀i. i < length gs =⇒ gs ! i < G′
    for gs j
  proof −
    consider condition7a gs jj | condition7b gs jj | condition7c gs jj
      by blast
    then show ?thesis
    proof (cases)
      case 1
      then have ∗: snd (cmdL7 jj gs) =
        [(gs ! 0, Stay),
         (enc-upd (gs ! 1) (k + jj) 0, Left),
         (gs ! 2, Stay)] @
         (map (λj. (if j = jj then 3 else gs ! (j + 3), Stay)) [0..<k]) @
         (map (λj. (gs ! (3 + k + j), Stay)) [0..<k])
        unfolding cmdL7-def by simp
      consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 ∗ k + 3
        using gs by linarith
      then show ?thesis
      proof (cases)
        case 1
        then show ?thesis
          using that by (simp add: goal2)
      next
        case 2
        then have is-code (gs ! 1)
          using 1 by blast
        moreover have k + jj < 2 ∗ k + 2
          using assms by simp
        moreover have 0 < G
          using G-ge-4 by simp
        ultimately have is-code (enc-upd (gs ! 1) (k + jj) 0)
          using enc-upd-is-code by simp
        then have is-code (cmdL7 jj gs [.] j)
          using ∗ 2 by simp
        then show ?thesis
          using G′-ge-G G′-def by simp
```

next
  case *3*
  then show *?thesis*
    using ∗ *gs* **by** *simp*
next
  case *4*
  then show *?thesis*
    using ∗ *gs G′-ge threeplus2k-2*[**where** *?a*=(*gs* ! *0*, *Stay*)] **by** *simp*
next
  case *5*
  then show *?thesis*
    using ∗ *gs G′-ge threeplus2k-3*[**where** *?a*=(*gs* ! *0*, *Stay*)] **by** *simp*
qed
next
  case *case2*: *2*
  then have ∗: *snd* (*cmdL7 jj gs*) =
    [(*gs* ! *0*, *Stay*),
     (*enc-upd* (*gs* ! *1*) (*k* + *jj*) *1*, *Left*),
     (*gs* ! *2*, *Stay*)] @
     (*map* (λ*j*. (*if j* = *jj then 0 else gs* ! (*j* + *3*), *Stay*)) [*0*..<*k*]) @
     (*map* (λ*j*. (*gs* ! (*3* + *k* + *j*), *Stay*)) [*0*..<*k*])
    unfolding *cmdL7-def* **by** *simp*
  consider *j* = *0* | *j* = *1* | *j* = *2* | *3* ≤ *j* ∧ *j* < *k* + *3* | *k* + *3* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
    using *gs* **by** *linarith*
  then show *?thesis*
  proof (*cases*)
    case *1*
    then show *?thesis*
      using *that* **by** (*simp add*: *goal2*)
  next
    case *2*
    then have *is-code* (*gs* ! *1*)
      using *case2* **by** *simp*
    moreover have *k* + *jj* < *2* ∗ *k* + *2*
      using *assms* **by** *simp*
    moreover have *1* < *G*
      using *G-ge-4* **by** *simp*
    ultimately have *is-code* (*enc-upd* (*gs* ! *1*) (*k* + *jj*) *1*)
      using *enc-upd-is-code* **by** *simp*
    then have *is-code* (*cmdL7 jj gs* [.] *j*)
      using ∗ *2* **by** *simp*
    then show *?thesis*
      using *G′-ge-G G′-def* **by** *simp*
  next
    case *3*
    then show *?thesis*
      using ∗ *gs* **by** *simp*
  next
    case *4*
    then show *?thesis*
      using ∗ *gs G′-ge threeplus2k-2*[**where** *?a*=(*gs* ! *0*, *Stay*)] **by** *simp*
  next
    case *5*
    then show *?thesis*
      using ∗ *gs G′-ge threeplus2k-3*[**where** *?a*=(*gs* ! *0*, *Stay*)] **by** *simp*
  qed
next
  case *case3*: *3*
  then have ∗: *snd* (*cmdL7 jj gs*) =
    [(*gs* ! *0*, *Stay*),
     (*gs* ! *1*, *Left*),
     (*gs* ! *2*, *Stay*)] @
     (*map* (λ*j*. (*gs* ! (*j* + *3*), *Stay*)) [*0*..<*k*]) @

```
      (map (λj. (gs ! (3 + k + j), Stay)) [0..<k])
      unfolding cmdL7-def by auto
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 ∗ k + 3
      using gs by linarith
    then show ?thesis
      using ∗ gs G′-ge threeplus2k-2[where ?a=(gs ! 0, Stay)] threeplus2k-3[where ?a=(gs ! 0, Stay)]
      by (cases) simp-all
   qed
  qed
  show ⋀gs. length gs = 2 ∗ k + 3 ⟹ [∗] (cmdL7 jj gs) ≤ 1
   using cmdL7-def by simp
qed

definition tmL67 :: nat ⇒ machine where
  tmL67 jj ≡ [cmdL7 jj]

lemma tmL67-tm:
  assumes jj < k
  shows turing-machine (2 ∗ k + 3) G′ (tmL67 jj)
  using assms G′-ge tmL67-def turing-command-cmdL7 by auto

definition tmL47 jj ≡ tmL46 jj ;; tmL67 jj

lemma tmL47-tm:
  assumes jj < k
  shows turing-machine (2 ∗ k + 3) G′ (tmL47 jj)
  using assms G′-ge tm-left-tm tmL46-tm tmL47-def tmL67-tm by simp
```

Next we are sweeping right again and perform the head movement for tape jj if this is a movement to the right. This works the same as the left movements in *cmdL7*.

```
definition cmdL8 :: nat ⇒ command where
  cmdL8 jj rs ≡
 (if rs ! 1 = □ then 1 else 0,
  if rs ! (3 + jj) = 2 ∧ enc-nth (rs ! 1) (k + jj) = 1 ∧ is-code (rs ! 1) then
   [(rs ! 0, Stay),
    (enc-upd (rs ! 1) (k + jj) 0, Right),
    (rs ! 2, Stay)] @
    (map (λj. (if j = jj then 3 else rs ! (j + 3), Stay)) [0..<k]) @
    (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
  else if rs ! (3 + jj) = 3 ∧ is-code (rs ! 1) then
   [(rs ! 0, Stay),
    (enc-upd (rs ! 1) (k + jj) 1, Right),
    (rs ! 2, Stay)] @
    (map (λj. (if j = jj then 2 else rs ! (j + 3), Stay)) [0..<k]) @
    (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
  else if rs ! 1 = 0 then
   [(rs ! 0, Stay),
    (rs ! 1, Stay),
    (rs ! 2, Stay)] @
    (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
    (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
  else
   [(rs ! 0, Stay),
    (rs ! 1, Right),
    (rs ! 2, Stay)] @
    (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
    (map (λj. (rs ! (3 + k + j), Stay)) [0..<k]))

abbreviation condition8a gs jj ≡
  gs ! (3 + jj) = 2 ∧ enc-nth (gs ! 1) (k + jj) = 1 ∧ is-code (gs ! 1)
abbreviation condition8b gs jj ≡
  ¬ condition8a gs jj ∧ gs ! (3 + jj) = 3 ∧ is-code (gs ! 1)
abbreviation condition8c gs jj ≡
```

¬ *condition8a gs jj* ∧ ¬ *condition8b gs jj* ∧ *gs* ! *1* = *0*
abbreviation *condition8d gs jj* ≡
¬ *condition8a gs jj* ∧ ¬ *condition8b gs jj* ∧ ¬ *condition8c gs jj*

lemma *turing-command-cmdL8*:
assumes *jj* < *k*
shows *turing-command* (*2* ∗ *k* + *3*) *1* *G′* (*cmdL8 jj*)
proof
show *length* ([!!] *cmdL8 jj gs*) = *length gs* **if** *length gs* = *2* ∗ *k* + *3* **for** *gs*
proof −
consider *condition8a gs jj* | *condition8b gs jj* | *condition8c gs jj* | *condition8d gs jj*
by *blast*
then show *?thesis*
unfolding *cmdL8-def* **using** *that* **by** (*cases*) *simp-all*
qed
show *goal2*: *0* < *2* ∗ *k* + *3* ⟹ *cmdL8 jj gs* [.] *0* = *gs* ! *0* **if** *length gs* = *2* ∗ *k* + *3* **for** *gs*
proof −
consider *condition8a gs jj* | *condition8b gs jj* | *condition8c gs jj* | *condition8d gs jj*
by *blast*
then show *?thesis*
unfolding *cmdL8-def* **using** *that* **by** (*cases*) *simp-all*
qed
show *cmdL8 jj gs* [.] *j* < *G′*
if *gs*: *j* < *length gs length gs* = *2* ∗ *k* + *3* ⋀*i*. *i* < *length gs* ⟹ *gs* ! *i* < *G′*
for *gs j*
proof −
consider *condition8a gs jj* | *condition8b gs jj* | *condition8c gs jj* | *condition8d gs jj*
by *blast*
then show *?thesis*
proof (*cases*)
case *1*
then have ∗: *snd* (*cmdL8 jj gs*) =
[(*gs* ! *0*, *Stay*),
(*enc-upd* (*gs* ! *1*) (*k* + *jj*) *0*, *Right*),
(*gs* ! *2*, *Stay*)] @
(*map* (λ*j*. (*if j* = *jj then 3 else gs* ! (*j* + *3*), *Stay*)) [*0*..<*k*]) @
(*map* (λ*j*. (*gs* ! (*3* + *k* + *j*), *Stay*)) [*0*..<*k*])
unfolding *cmdL8-def* **by** *simp*
consider *j* = *0* | *j* = *1* | *j* = *2* | *3* ≤ *j* ∧ *j* < *k* + *3* | *k* + *3* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
using *gs* **by** *linarith*
then show *?thesis*
proof (*cases*)
case *1*
then show *?thesis*
using *that* **by** (*simp add*: *goal2*)
next
case *2*
then have *is-code* (*gs* ! *1*)
using *1* **by** *blast*
moreover have *k* + *jj* < *2* ∗ *k* + *2*
using *assms* **by** *simp*
moreover have *0* < *G*
using *G-ge-4* **by** *simp*
ultimately have *is-code* (*enc-upd* (*gs* ! *1*) (*k* + *jj*) *0*)
using *enc-upd-is-code* **by** *simp*
then have *is-code* (*cmdL8 jj gs* [.] *j*)
using ∗ *2* **by** *simp*
then show *?thesis*
using *G′-ge-G G′-def* **by** *simp*
next
case *3*
then show *?thesis*
using ∗ *gs* **by** *simp*

```
    next
      case 4
      then show ?thesis
        using * gs G'-ge threeplus2k-2[where ?a=(gs ! 0, Stay)] by simp
    next
      case 5
      then show ?thesis
        using * gs G'-ge threeplus2k-3[where ?a=(gs ! 0, Stay)] by simp
    qed
  next
    case case2: 2
    then have *: snd (cmdL8 jj gs) =
      [(gs ! 0, Stay),
       (enc-upd (gs ! 1) (k + jj) 1, Right),
       (gs ! 2, Stay)] @
       (map (λj. (if j = jj then 2 else gs ! (j + 3), Stay)) [0..<k]) @
       (map (λj. (gs ! (3 + k + j), Stay)) [0..<k])
      unfolding cmdL8-def by simp
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
      using gs by linarith
    then show ?thesis
    proof (cases)
      case 1
      then show ?thesis
        using that by (simp add: goal2)
    next
      case 2
      then have is-code (gs ! 1)
        using case2 by simp
      moreover have k + jj < 2 * k + 2
        using assms by simp
      moreover have 1 < G
        using G-ge-4 by simp
      ultimately have is-code (enc-upd (gs ! 1) (k + jj) 1)
        using enc-upd-is-code by simp
      then have is-code (cmdL8 jj gs [.] j)
        using * 2 by simp
      then show ?thesis
        using G'-ge-G G'-def by simp
    next
      case 3
      then show ?thesis
        using * gs by simp
    next
      case 4
      then show ?thesis
        using * gs G'-ge threeplus2k-2[where ?a=(gs ! 0, Stay)] by simp
    next
      case 5
      then show ?thesis
        using * gs G'-ge threeplus2k-3[where ?a=(gs ! 0, Stay)] by simp
    qed
  next
    case 3
    then have *: snd (cmdL8 jj gs) =
      [(gs ! 0, Stay),
       (gs ! 1, Stay),
       (gs ! 2, Stay)] @
       (map (λj. (gs ! (j + 3), Stay)) [0..<k]) @
       (map (λj. (gs ! (3 + k + j), Stay)) [0..<k])
      unfolding cmdL8-def by auto
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
      using gs by linarith
```

      then show *?thesis*
        using $*$ *gs* G'*-ge threeplus2k-2*[**where** *?a*=(*gs* ! *0*, *Stay*)] *threeplus2k-3*[**where** *?a*=(*gs* ! *0*, *Stay*)]
        by (*cases*) *simp-all*
    next
      case *4*
      then have $*$: *snd* (*cmdL8 jj gs*) =
        [(*gs* ! *0*, *Stay*),
         (*gs* ! *1*, *Right*),
         (*gs* ! *2*, *Stay*)] @
         (*map* (λ*j*. (*gs* ! (*j* + *3*), *Stay*)) [*0*..<*k*]) @
         (*map* (λ*j*. (*gs* ! (*3* + *k* + *j*), *Stay*)) [*0*..<*k*])
        unfolding *cmdL8-def* **by** *auto*
      consider $j = 0 \mid j = 1 \mid j = 2 \mid 3 \leq j \land j < k + 3 \mid k + 3 \leq j \land j < 2 * k + 3$
        using *gs* **by** *linarith*
      then show *?thesis*
        using $*$ *gs* G'*-ge threeplus2k-2*[**where** *?a*=(*gs* ! *0*, *Stay*)] *threeplus2k-3*[**where** *?a*=(*gs* ! *0*, *Stay*)]
        by (*cases*) *simp-all*
    qed
  qed
  show $\bigwedge$*gs*. *length gs* = *2* $*$ *k* + *3* $\Longrightarrow$ [$*$] (*cmdL8 jj gs*) $\leq$ *1*
    using *cmdL8-def* **by** *simp*
qed

definition *tmL78* :: *nat* $\Rightarrow$ *machine* **where**
  tmL78 jj $\equiv$ [*cmdL8 jj*]

lemma *tmL78-tm*:
  assumes *jj* < *k*
  shows *turing-machine* (*2* $*$ *k* + *3*) G' (*tmL78 jj*)
  using *assms* G'*-ge tmL78-def turing-command-cmdL8* **by** *auto*

definition *tmL48 jj* $\equiv$ *tmL47 jj* ;; *tmL78 jj*

lemma *tmL48-tm*:
  assumes *jj* < *k*
  shows *turing-machine* (*2* $*$ *k* + *3*) G' (*tmL48 jj*)
  using *assms* G'*-ge tm-left-tm tmL47-tm tmL48-def tmL78-tm* **by** *simp*

The last command in the command sequence is moving back to the beginning of the output tape.

definition *tmL49 jj* $\equiv$ *tmL48 jj* ;; *tm-left-until1*

The Turing machine *tmL49 jj* is then repeated for the parameters $jj = 0, \ldots, k-1$ in order to simulate the actions of M on all tapes.

lemma *tmL49-tm*: *jj* < *k* $\Longrightarrow$ *turing-machine* (*2* $*$ *k* + *3*) G' (*tmL49 jj*)
  using *tmL48-tm tmL49-def tmL1-def tm-left-until1-tm* **by** *simp*

fun *tmL49-upt* :: *nat* $\Rightarrow$ *machine* **where**
  tmL49-upt 0 = [] |
  tmL49-upt (*Suc j*) = *tmL49-upt j* ;; *tmL49 j*

lemma *tmL49-upt-tm*:
  assumes *j* $\leq$ *k*
  shows *turing-machine* (*2* $*$ *k* + *3*) G' (*tmL49-upt j*)
  using *assms*
proof (*induction j*)
  case *0*
  then show *?case*
    using G'*-ge*(*1*) *turing-machine-def* **by** *simp*
next
  case (*Suc j*)
  then show *?case*
    using *assms tmL49-tm* **by** *simp*
qed

definition *tmL9* ≡ *tmL4* ;; *tmL49-upt k*

lemma *tmL9-tm*: *turing-machine* $(2 * k + 3)$ G' *tmL9*
  unfolding *tmL9-def* **using** *tmL49-upt-tm tmL4-tm* **by** *simp*

At this point in the iteration we have completed one more step in the execution of M. We mark this be setting one more counter flag, namely the one in the leftmost code symbol where the flag is still unset. To find the first unset counter flag, we reuse *tmC*.

definition *tmL10* ≡ *tmL9* ;; *tmC*

lemma *tmL10-tm*: *turing-machine* $(2 * k + 3)$ G' *tmL10*
  unfolding *tmL10-def* **using** *tmL9-tm tmC-tm* **by** *simp*

Then we set the counter flag, unless we have reached a blank symbol.

```
definition cmdL11 :: command where
  cmdL11 rs ≡
   (1,
    [(rs ! 0, Stay),
     if is-code (rs ! 1) then (enc-upd (rs ! 1) (2 ∗ k) 1, Stay) else (rs ! 1, Stay),
     (rs ! 2, Stay)] @
     (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
     (map (λj. (rs ! (3 + k + j), Stay)) [0..<k]))

lemma turing-command-cmdL11: turing-command (2 ∗ k + 3) 1 G' cmdL11
proof
  show length gs = 2 ∗ k + 3 =⇒ length ([!!] cmdL11 gs) = length gs for gs
    unfolding cmdL11-def by (cases gs ! 1 = 0) simp-all
  show goal2: length gs = 2 ∗ k + 3 =⇒ 0 < 2 ∗ k + 3 =⇒ cmdL11 gs [.] 0 = gs ! 0 for gs
    unfolding cmdL11-def by (cases gs ! 1 = 0) simp-all
  show cmdL11 gs [.] j < G'
    if length gs = 2 ∗ k + 3 ⋀i. i < length gs =⇒ gs ! i < G' j < length gs
    for gs j
  proof −
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 ∗ k + 3
      using ‹length gs = 2 ∗ k + 3› ‹j < length gs› by linarith
    then show ?thesis
    proof (cases)
      case 1
      then show ?thesis
        using that goal2 by simp
    next
      case 2
      show ?thesis
      proof (cases is-code (gs ! 1))
        case True
        then have ∗: cmdL11 gs [.] j = enc-upd (gs ! 1) (2 ∗ k) 1
          using 2 cmdL11-def by simp
        then have is-code (cmdL11 gs [.] j)
          using enc-upd-is-code[OF True] G-ge-4 by simp
        then show ?thesis
          using G'-def by simp
      next
        case False
        then show ?thesis
          using that cmdL11-def 2 by auto
      qed
    next
      case 3
      then show ?thesis
        using that cmdL11-def by simp
    next
      case 4
```

then show *?thesis*
using *that cmdL11-def threeplus2k-2*[*OF 4*, *of* (*gs* ! *0*, *Stay*)] **by** *simp*
next
case *5*
then show *?thesis*
using *that cmdL11-def threeplus2k-3*[*OF 5*, *of* (*gs* ! *0*, *Stay*)] **by** *simp*
qed
qed
show $\bigwedge gs.\ length\ gs = 2 * k + 3 \Longrightarrow [*]\ (cmdL11\ gs) \leq 1$
using *cmdL11-def* **by** *simp*
qed

definition *tmL11* ≡ *tmL10* ;; [*cmdL11*]

lemma *tmL1011-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′* [*cmdL11*]
using *cmdL11-def turing-command-cmdL11 G′-ge* **by** *auto*

lemma *tmL11-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tmL11*
using *tmL11-def tmL1011-tm G′-ge tmL10-tm* **by** *simp*

And we move back to the beginning of the output tape again.

definition *tmL12* ≡ *tmL11* ;; *tm-left-until1*

lemma *tmL12-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tmL12*
using *tmL11-tm tmL12-def tm-left-until1-tm* **by** *simp*

Now, at the end of the iteration we set the memorization tapes $3, \dots, 2k+2$, that is, all but the one memorizing the state of M, to 0. This makes for a simpler loop invariant than having the leftover symbols there.

definition *tmL13* ≡ *tmL12* ;; *tm-write-many* {*3*..<*2* ∗ *k* + *3*} *0*

lemma *tmL13-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tmL13*
unfolding *tmL13-def* **using** *tmL12-tm tm-write-many-tm k-ge-2 G′-ge*(*1*) **by** *simp*

This is the entire loop. It terminates once there are no unset counter flags anymore.

definition *tmLoop* ≡ *WHILE tmC* ; *λrs. rs* ! *1* > □ *DO tmL13 DONE*

lemma *tmLoop-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tmLoop*
using *tmLoop-def turing-machine-loop-turing-machine tmC-tm tmL13-tm* **by** *simp*

definition *tm10* ≡ *tm9* ;; *tmLoop*

lemma *tm10-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′ tm10*
using *tm10-def tmLoop-tm tm9-tm* **by** *simp*

Cleaning up the output

Now the simulation proper has ended, but the output tape does not yet look quite like the output tape of M. It remains to extract the component 1 from all the code symbols on the output tape. The simulator does this while sweeping left. Formally, "extracting component 1" means this:

abbreviation *ec1* :: *symbol* ⇒ *symbol* **where**
ec1 h ≡ *if is-code h then enc-nth h 1 else* □

lemma *ec1*: *ec1 h* < *G′* **if** *h* < *G′* **for** *h*
proof (*cases is-code h*)
case *True*
then show *?thesis*
using *enc-nth-less G′-ge-G* **by** *fastforce*
next
case *False*
then show *?thesis*
using *that* **by** *auto*

qed

definition *tm11* ≡ *tm10* ;; *tm-ltrans-until 1 1 StartSym ec1*

lemma *tm11-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′* *tm11*
proof −
  have *turing-machine* (*2* ∗ *k* + *3*) *G′* (*tm-ltrans-until 1 1 StartSym ec1*)
    using *G′-ge ec1* **by** (*intro tm-ltrans-until-tm*) *simp-all*
  then show *?thesis*
    using *tm10-tm tm11-def* **by** *simp*
qed

The previous TM, *tm-ltrans-until 1 1* $\{y.\ y < G^{2*k+2} + 2 \wedge 1 < y \wedge dec\ y\ !\ (2*k+1) = 1\}$ (λ*h*. *if is-code h then enc-nth h 1 else 0*), halts as soon as it encounters a code symbol with the start flag set, without applying the extraction function. Applying the function to this final code symbol, which is at the leftmost cell of the tape, is the last step of the simulator machine.

definition *tm12* ≡ *tm11* ;; *tm-rtrans 1 ec1*

lemma *tm12-tm*: *turing-machine* (*2* ∗ *k* + *3*) *G′* *tm12*
  unfolding *tm12-def* **using** *tm-rtrans-tm tm11-tm ec1 G′-ge* **by** *simp*

5.3.3 Semantics of the Turing machine

This section establishes not only the configurations of the simulator but also the traces. For every Turing machine and command defined in the previous subsection, there will be a corresponding trace and a tape list representing the simulator's configuration after the command or TM has been applied.

For most of the time, the simulator's output tape will have no start symbol, and so the next definition will be more suited to describing it than the customary *contents*.

definition *contents′* :: *symbol list* ⇒ (*nat* ⇒ *symbol*) **where**
  contents′ ys x ≡ *if x* < *length ys then ys* ! *x else 0*

lemma *contents′-eqI*:
  assumes ⋀*x*. *x* < *length ys* ⟹ *zs x* = *ys* ! *x*
    and ⋀*x*. *x* ≥ *length ys* ⟹ *zs x* = *0*
  shows *zs* = *contents′ ys*
  using *contents′-def assms* **by** *auto*

lemma *contents-contents′*: ⌊*ys*⌋ = *contents′* (*1* # *ys*)
  using *contents-def contents′-def* **by** *auto*

lemma *contents′-at-ge*:
  assumes *i* ≥ *length ys*
  shows *contents′ ys i* = *0*
  using *assms contents′-def* **by** *simp*

In the following context *zs* represents the input for M and hence for the simulator.

context
  fixes *zs* :: *symbol list*
  assumes *zs*: *bit-symbols zs*
begin

lemma *zs-less-G*: ∀ *i*<*length zs*. *zs* ! *i* < *G*
  using *zs G-ge-4* **by** *auto*

lemma *zs-proper*: *proper-symbols zs*
  using *zs* **by** *auto*

abbreviation *n* ≡ *length zs*

abbreviation *TT* ≡ *Suc* (*fmt n*)

Initializing the simulator's tapes

definition *tps0* :: *tape list* **where**
tps0 ≡
[(⌊*zs*⌋, *0*),
(⌊[]⌋, *0*)] @
replicate (*2* ∗ *k* + *1*) (⌈▷⌉)

lemma *tps0-start-config*: *start-config* (*2* ∗ *k* + *3*) *zs* = (*0*, *tps0*)
unfolding *tps0-def contents-def onesie-def start-config-def* **by** *auto*

definition *tps1* :: *tape list* **where**
tps1 ≡
[(⌊*zs*⌋, *1*),
(⌊*replicate* (*fmt n*) *3*⌋, *1*)] @
replicate (*2* ∗ *k* + *1*) (⌈▷⌉)

definition *es1* ≡ *es-fmt n*

lemma *tm1*: *traces tm1 tps0 es1 tps1*
proof −
let *?tps* = *replicate* (*2* ∗ *k* + *1*) (⌈▷⌉)
have *1*: *tps0* = *start-tapes-2 zs* @ *?tps*
using *contents-def tps0-def start-tapes-2-def* **by** *auto*
have *2*: *tps1* = *one-tapes-2 zs* (*fmt n*) @ *?tps*
using *contents-def tps1-def one-tapes-2-def* **by** *simp*
have *length* (*start-tapes-2 zs*) = *2*
using *start-tapes-2-def* **by** *simp*
moreover have *traces tm-fmt* (*start-tapes-2 zs*) (*es-fmt n*) (*one-tapes-2 zs* (*fmt n*))
using *fmt zs* **by** *fastforce*
moreover have *turing-machine 2 G′ tm-fmt* **using** *fmt*(*1*) **.**
ultimately have
traces (*append-tapes 2* (*2* + *length ?tps*) *tm-fmt*) (*start-tapes-2 zs* @ *?tps*) (*es-fmt n*) (*one-tapes-2 zs* (*fmt n*) @ *?tps*)
using *traces-append-tapes* **by** *blast*
then have
traces (*append-tapes 2* (*2* ∗ *k* + *3*) *tm-fmt*) (*start-tapes-2 zs* @ *?tps*) (*es-fmt n*) (*one-tapes-2 zs* (*fmt n*) @ *?tps*)
by (*simp add*: *numeral-3-eq-3*)
then have *traces* (*append-tapes 2* (*2* ∗ *k* + *3*) *tm-fmt*) *tps0* (*es-fmt n*) *tps1*
using *1 2* **by** *simp*
then show *?thesis*
using *tm1-def es1-def* **by** *simp*
qed

definition *es1-2* ≡ *map* (λ*i*. (*1*, *1* + *Suc i*)) [*0*..<*fmt n*] @ [(*1*, *1* + *fmt n*)]

definition *es2* ≡ *es1* @ *es1-2*

lemma *len-es2*: *length es2* = *length* (*es-fmt n*) + *TT*
using *es2-def es1-def* **by** (*simp add*: *es1-2-def*)

definition *tps2* :: *tape list* **where**
tps2 ≡
[(⌊*zs*⌋, *1*),
(⌊*replicate* (*fmt n*) (*enc* (*replicate k 0* @ *replicate k 0* @ [*0*, *0*]))⌋, *TT*)] @
replicate (*2* ∗ *k* + *1*) (⌈▷⌉)

lemma *tm2*: *traces tm2 tps0 es2 tps2*
unfolding *tm2-def es2-def*
proof (*rule traces-sequential*)
show *traces tm1 tps0 es1 tps1* **using** *tm1* **.**
show *traces tm1-2 tps1 es1-2 tps2*
unfolding *tm1-2-def es1-2-def*

```
  proof (rule traces-tm-const-until-11I[where ?n=fmt n and ?G=G'])
    show 1 < length tps1
      using tps1-def by simp
    show enc (replicate k 0 @ replicate k 0 @ [0, 0]) < G'
      using G-ge-4 by (intro enc-less-G') simp-all
    show rneigh (tps1 ! 1) {0} (fmt n)
      using tps1-def contents-def by (intro rneighI) simp-all
    show map (λi. (1, 1 + Suc i)) [0..<fmt n] @ [(1, 1 + fmt n)] =
        map (λi. (tps1 :#: 0, tps1 :#: 1 + Suc i)) [0..<fmt n] @ [(tps1 :#: 0, tps1 :#: 1 + fmt n)]
      using tps1-def contents-def by simp
    show tps2 =
        tps1 [1 := constplant (tps1 ! 1) (enc (replicate k 0 @ replicate k 0 @ [0, 0])) (fmt n)]
      using tps2-def tps1-def contents-def constplant by auto
  qed
qed

definition es3 ≡ es2 @ map (λi. (1, i)) (rev [0..<TT]) @ [(1, 0)]

definition tps3 :: tape list where
  tps3 ≡
    [(⌊zs⌋, 1),
     (⌊replicate (fmt n) (enc (replicate k 0 @ replicate k 0 @ [0, 0]))⌋, 0)] @
    replicate (2 * k + 1) (⌈▷⌉)

lemma tm3: traces tm3 tps0 es3 tps3
  unfolding tm3-def es3-def
proof (rule traces-sequential)
  show traces tm2 tps0 es2 tps2 using tm2 .
  show traces (tm-start 1) tps2 (map (λi. (1, i)) (rev [0..<TT]) @ [(1, 0)]) tps3
    using tps3-def tps2-def enc-greater by (intro traces-tm-start-1I) simp-all
qed

definition es4 ≡ es3 @ [(1, 0)]

lemma len-es4: length es4 = length (es-fmt n) + 2 * TT + 2
  using es4-def es3-def len-es2 by simp

definition tps4 :: tape list where
  tps4 ≡
    [(⌊zs⌋, 1),
     (contents'
       ((enc (replicate k 1 @ replicate k 1 @ [0, 1])) #
        replicate (fmt n) (enc (replicate k 0 @ replicate k 0 @ [0, 0]))), 0)] @
    replicate (2 * k + 1) (⌈▷⌉)

lemma tm4: traces tm4 tps0 es4 tps4
  unfolding tm4-def es4-def
proof (rule traces-sequential)
  show traces tm3 tps0 es3 tps3 using tm3 .
  show traces (tm-write 1 (enc (replicate k 1 @ replicate k 1 @ [0, 1]))) tps3 [(1, 0)] tps4
  proof (rule traces-tm-write-1I)
    show 1 < length tps3
      using tps3-def by simp
    show [(1, 0)] = [(tps3 :#: 0, tps3 :#: 1)]
      using tps3-def by simp
    show tps4 = tps3[1 := tps3 ! 1 |:=| enc (replicate k 1 @ replicate k 1 @ [0, 1])]
      using tps3-def tps4-def contents'-def contents-contents' by auto
  qed
qed

definition es5 ≡ es4 @ [(1, 1)]

lemma len-es5: length es5 = length (es-fmt n) + 2 * TT + 3
```

using *es5-def len-es4* **by** *simp*

definition *tps5* :: *tape list* **where**
tps5 ≡
[(⌊*zs*⌋, *1*),
(*contents'*
((*enc* (*replicate k 1* @ *replicate k 1* @ [*0*, *1*])) #
replicate (*fmt n*) (*enc* (*replicate k 0* @ *replicate k 0* @ [*0*, *0*]))), *1*)] @
replicate (*2* ∗ *k* + *1*) (⌈▷⌉)

lemma *tm5*: *traces tm5 tps0 es5 tps5*
unfolding *tm5-def es5-def*
proof (*rule traces-sequential*)
show *traces tm4 tps0 es4 tps4*
using *tm4* **.**
show *traces* (*tm-right 1*) *tps4* [(*1*, *1*)] *tps5*
using *tps4-def tps5-def* **by** (*intro traces-tm-right-1I*) *simp-all*
qed

Since the simulator simulates M on *zs*, its tape contents are typically described in terms of configurations of M. So the following definition is actually more like an abbreviation.

definition *exec t* ≡ *execute M* (*start-config k zs*) *t*

lemma *exec-pos-less-TT*:
assumes *j* < *k*
shows *exec t* <#> *j* < *TT*
proof −
have *exec t* <#> *j* ≤ *T' n*
using *head-pos-le-time-bound*[*OF tm-M time-bound-T' zs assms*] *exec-def* **by** *simp*
then show *?thesis*
by (*meson fmt*(*4*) *le-imp-less-Suc le-trans*)
qed

lemma *tps-ge-TT-0*:
assumes *i* ≥ *TT*
shows (*exec t* <:> *1*) *i* = *0*
proof (*induction t*)
case *0*
have *exec 0* = *start-config k zs*
using *exec-def* **by** *simp*
then show *?case*
using *start-config1 assms tm-M turing-machine-def* **by** *simp*
next
case (*Suc t*)
have ∗: *exec* (*Suc t*) = *exe M* (*exec t*)
using *exec-def* **by** *simp*
show *?case*
proof (*cases fst* (*exec t*) ≥ *length M*)
case *True*
then have *exec* (*Suc t*) = *exec t*
using ∗ *exe-def* **by** *simp*
then show *?thesis*
using *Suc* **by** *simp*
next
case *False*
then have *exec* (*Suc t*) <:> *1* = *sem* (*M* ! (*fst* (*exec t*))) (*exec t*) <:> *1*
(**is** - = *sem ?cmd ?cfg* <:> *1*)
using *exe-lt-length* ∗ **by** *simp*
also have ... = *fst* (*map* (λ(*a*, *tp*). *act a tp*) (*zip* (*snd* (*?cmd* (*read* (*snd ?cfg*)))) (*snd ?cfg*)) ! *1*)
using *sem'* **by** *simp*
also have ... = *fst* (*act* (*snd* (*?cmd* (*read* (*snd ?cfg*))) ! *1*) (*snd ?cfg* ! *1*))
(**is** - = *fst* (*act ?h* (*snd ?cfg* ! *1*)))
proof −

```
      have ||?cfg|| = k
        using exec-def tm-M execute-num-tapes[OF tm-M] start-config-length turing-machine-def
        by simp
      moreover from this have length (snd (?cmd (read (snd ?cfg)))) = k
        by (metis False turing-commandD(1) linorder-not-less read-length tm-M turing-machineD(3))
      moreover have k > 1
         using k-ge-2 by simp
      ultimately show ?thesis
        by simp
    qed
    finally have exec (Suc t) <:> 1 = fst (act ?h (?cfg <!> 1)) .
    moreover have i ≠ snd (?cfg <!> 1)
      using assms by (metis Suc-1 exec-pos-less-TT lessI less-irrefl less-le-trans tm-M turing-machine-def)
    ultimately have (exec (Suc t) <:> 1) i = fst (?cfg <!> 1) i
      using act-changes-at-most-pos by (metis prod.collapse)
    then show ?thesis
      using Suc.IH by simp
  qed
qed
```

The next definition is central to how we describe the simulator's output tape. The basic idea is that it describes the tape during the simulation of the t-th step of executing M on input zs. Recall that TT is the length of the formatted part on the simulator's output tape. The i-th cell of the output tape contains: (1) k symbols corresponding to the symbols in the i-th cell of the k tapes of M after t steps; (2) k flags indicating which of the k tape heads is in position i; (3) a unary counter representing the number t; (4) a flag indicating whether $i = 0$. This is the situation at the beginning of the t-iteration of the simulator's main loop. During this iteration the tape changes slightly: some symbols and head positions may already represent the situation after $t + 1$ steps of M, that is, the t-th step has been partially simulated already.

To account for this, there is the xs parameter. It is meant to be set to a list of k pairs. Let the j-th element of this list be (a, b). on M's tape j has already been simulated. In other words, the output tape reflects the situation after $t + a$ steps. Likewise b will be either None or 0 or 1. If it is 0 or 1, it means the flag represents the head position of tape j after $t + b$ steps. If it is None, it means that all head flags for tape k are currently zero, representing a "tape without head". This situation occurs every time the simulator has cleared the head flag representing the position after t steps, bus has not set the flag for the head position after $t + 1$ steps yet.

Therefore, at the beginning of the t-t iteration of the simulator's loop xs consists of k pairs $(0, 0)$. During the iteration every pair will eventually become $(0, 0)$.

```
definition zip-cont :: nat ⇒ (nat × nat option) list ⇒ (nat ⇒ symbol) where
  zip-cont t xs i ≡
    if i < TT then enc
     (map (λj. (exec (t + fst (xs ! j)) <:> j) i) [0..<k] @
      map (λj. case snd (xs ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k] @
      [if i < t then 1 else 0,
       if i = 0 then 1 else 0])
    else 0
```

Some auxiliary lemmas for accessing elements of lists of certain shape:

```
lemma less-k-nth: j < k ⟹ (map f1 [0..<k] @ map f2 [0..<k] @ [a, b]) ! j = f1 j
  by (simp add: nth-append)

lemma less-2k-nth: k ≤ j ⟹ j < 2 * k ⟹ (map f1 [0..<k] @ map f2 [0..<k] @ [a, b]) ! j = f2 (j − k)
proof −
  assume a: k ≤ j j < 2 * k
  have b: (map f1 [0..<k] @ map f2 [0..<k]) ! (k + l) = f2 l if l < k for l
    by (simp add: nth-append that)
  have (map f1 [0..<k] @ map f2 [0..<k]) ! j = f2 (j − k)
  proof −
    obtain l where l: l < k k + l = j
      using a le-Suc-ex by auto
    then have (map f1 [0..<k] @ map f2 [0..<k]) ! (k + l) = f2 l
      using b by auto
```

```
    with l show ?thesis
      by auto
  qed
  moreover have (map f1 [0..<k] @ map f2 [0..<k] @ [a, b]) = (map f1 [0..<k] @ map f2 [0..<k]) @ [a, b]
    by simp
  moreover have length (map f1 [0..<k] @ map f2 [0..<k]) = 2 * k
    by simp
  ultimately show ?thesis
    using a by (metis nth-append)
qed

lemma twok-nth: (map f1 [0..<k] @ map f2 [0..<k] @ [a, b]) ! (2 * k) = a
proof −
  have map f1 [0..<k] @ map f2 [0..<k] @ [a, b] = (map f1 [0..<k] @ map f2 [0..<k]) @ [a, b]
    by simp
  moreover have length (map f1 [0..<k] @ map f2 [0..<k]) = 2 * k
    by simp
  ultimately show ?thesis
    by (metis nth-append-length)
qed

lemma twok1-nth: (map f1 [0..<k] @ map f2 [0..<k] @ [a, b]) ! (2 * k + 1) = b
proof −
  have map f1 [0..<k] @ map f2 [0..<k] @ [a, b] = (map f1 [0..<k] @ map f2 [0..<k]) @ [a, b]
    by simp
  moreover have length (map f1 [0..<k] @ map f2 [0..<k]) = 2 * k
    by simp
  ultimately show ?thesis
    by (metis One-nat-def nth-Cons-0 nth-Cons-Suc nth-append-length-plus)
qed

lemma zip-cont-less-G:
  assumes i < TT
  shows ∀x∈set (map (λj. (exec (t + fst (xs ! j)) <:> j) i) [0..<k] @
      map (λj. case snd (xs ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k] @
      [if i < t then 1 else 0,
       if i = 0 then 1 else 0]). x < G
    (is ∀x∈set(?us @ ?vs @ [?a, ?b]). x < G)
proof −
  let ?ys = ?us @ ?vs @ [?a, ?b]
  let ?f1 = (λj. (exec (t + fst (xs ! j)) <:> j) i)
  let ?f2 = (λj. case snd (xs ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0)
  have ?ys ! j < G if j < 2 * k + 2 for j
  proof −
    consider j < k | k ≤ j ∧ j < 2 * k | j = 2 * k | j = 2 * k + 1
      using ‹j < 2 * k + 2› by linarith
    then show ?thesis
    proof (cases)
      case 1
      then have ?us ! j = (execute M (start-config k zs) (t + fst (xs ! j)) <:> j) i
        using exec-def by simp
      then show ?thesis
        using tape-alphabet[OF tm-M] zs-less-G by (simp add: 1 nth-append)
    next
      case 2
      then have ?ys ! j = (case snd (xs ! (j−k)) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> (j−k) then
1 else 0)
        by (simp add: less-2k-nth)
      then have ?ys ! j ≤ 1
        by (cases snd (xs ! (j − k))) auto
      then show ?thesis
        using G-ge-4 by simp
    next
```

```
      case 3
      then have ?ys ! j ≤ 1
        by (simp add: twok-nth)
      then show ?thesis
        using G-ge-4 by simp
    next
      case 4
      then have ?ys ! j = (if i = 0 then 1 else 0)
        using twok1-nth[of ?f1 ?f2 ?a ?b] by simp
      then show ?thesis
        using G-ge-4 by simp
    qed
  qed
  moreover have length ?ys = 2 ∗ k + 2
    by simp
  ultimately show ∀ x∈set ?ys. x < G
    by (metis (no-types, lifting) in-set-conv-nth)
qed

lemma dec-zip-cont:
  assumes i < TT
  shows dec (zip-cont t xs i) =
     (map (λj. (exec (t + fst (xs ! j)) <:> j) i) [0..<k] @
      map (λj. case snd (xs ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k] @
      [if i < t then 1 else 0,
       if i = 0 then 1 else 0])
    (is - = ?ys)
proof −
  have ∀ x∈set ?ys. x < G
    using zip-cont-less-G[OF assms] by simp
  moreover have len: length ?ys = 2 ∗ k + 2
    by simp
  ultimately have dec (enc ?ys) = ?ys
    using dec-enc by simp
  then show ?thesis
    using zip-cont-def assms by simp
qed

lemma zip-cont-gt-1:
  assumes i < TT
  shows zip-cont t xs i > 1
  using assms enc-greater zip-cont-def by simp

lemma zip-cont-less:
  assumes i < TT
  shows zip-cont t xs i < G ^ (2 ∗ k + 2) + 2
  using assms enc-less zip-cont-less-G zip-cont-def by simp

lemma zip-cont-eq-0:
  assumes i ≥ TT
  shows zip-cont t xs i = 0
  using assms zip-cont-def by simp

lemma dec-zip-cont-less-k:
  assumes i < TT and j < k
  shows dec (zip-cont t xs i) ! j = (exec (t + fst (xs ! j)) <:> j) i
  using dec-zip-cont[OF assms(1)] using assms(2) less-k-nth by (simp add: less-k-nth)

lemma dec-zip-cont-less-2k:
  assumes i < TT and j ≥ k and j < 2 ∗ k
  shows dec (zip-cont t xs i) ! j =
    (case snd (xs ! (j − k)) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> (j − k) then 1 else 0)
  using dec-zip-cont[OF assms(1)] assms(2,3) by (simp add: less-2k-nth)
```

```
lemma dec-zip-cont-2k:
  assumes i < TT
  shows dec (zip-cont t xs i) ! (2 ∗ k) = (if i < t then 1 else 0)
  using dec-zip-cont[OF assms(1)] by (simp add: twok-nth)

lemma dec-zip-cont-2k1:
  assumes i < TT
  shows dec (zip-cont t xs i) ! (2 ∗ k + 1) = (if i = 0 then 1 else 0)
  using dec-zip-cont[OF assms(1)] twok1-nth by force

lemma zip-cont-eqI:
  assumes i < TT
    and ⋀j. j < k =⇒ (exec (t + fst (xs ! j)) <:> j) i = (exec (t + fst (xs′ ! j)) <:> j) i
    and ⋀j. j < k =⇒
      (case snd (xs ! j) of None ⇒ (0::nat) | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) =
      (case snd (xs′ ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0)
  shows zip-cont t xs i = zip-cont t xs′ i
proof −
  have ∗: map (λj. case snd (xs ! j) of None ⇒ (0::nat) | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0)
[0..<k] =
      map (λj. case snd (xs′ ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k]
    using assms(3) by simp
  have zip-cont t xs i = enc
     (map (λj. (exec (t + fst (xs ! j)) <:> j) i) [0..<k] @
      map (λj. case snd (xs ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k] @
      [if i < t then 1 else 0,
       if i = 0 then 1 else 0])
    using assms zip-cont-def by simp
  also have ... = enc
     (map (λj. (exec (t + fst (xs′ ! j)) <:> j) i) [0..<k] @
      map (λj. case snd (xs ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k] @
      [if i < t then 1 else 0,
       if i = 0 then 1 else 0])
    using assms(2) by (smt (verit) atLeastLessThan-iff map-eq-conv set-upt)
  also have ... = enc
     (map (λj. (exec (t + fst (xs′ ! j)) <:> j) i) [0..<k] @
      map (λj. case snd (xs′ ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k] @
      [if i < t then 1 else 0,
       if i = 0 then 1 else 0])
    using ∗ by metis
  finally show ?thesis
    using zip-cont-def by auto
qed

lemma zip-cont-nth-eqI:
  assumes i < TT
    and ⋀j. j < k =⇒ (exec (t + fst (xs ! j)) <:> j) i = (exec (t + fst (xs′ ! j)) <:> j) i
    and ⋀j. j < k =⇒ snd (xs ! j) = snd (xs′ ! j)
  shows zip-cont t xs i = zip-cont t xs′ i
  using assms zip-cont-eqI by presburger

lemma begin-tape-zip-cont:
  begin-tape {y. y < G ^ (2 ∗ k + 2) + 2 ∧ 1 < y ∧ dec y ! (2 ∗ k + 1) = 1} (zip-cont t xs, i)
    (is begin-tape ?ys -)
proof −
  let ?y = zip-cont t xs 0
  have ?y ∈ ?ys
  proof −
    have ∗: ?y = enc
      (map (λj. (exec (t + fst (xs ! j)) <:> j) 0) [0..<k] @
       map (λj. case snd (xs ! j) of None ⇒ 0 | Some d ⇒ if 0 = exec (t + d) <#> j then 1 else 0) [0..<k] @
       [if 0 < t then 1 else 0, 1])
```

(**is** - = *enc ?z*)
using *zip-cont-def* **by** *simp*
then have *1*: *?y* > *1*
using *enc-greater* **by** *simp*
have *?z* ! (*2* ∗ *k* + *1*) = *1*
using *twok1-nth* **by** *fast*
then have *2*: *dec ?y* ! (*2* ∗ *k* + *1*) = *1*
using *dec-zip-cont* **by** *simp*
have *?y* < *G* ^ (*2* ∗ *k* + *2*) + *2*
using *enc-less* ∗ *zip-cont-less-G*[*of 0*] **by** *simp*
then show *?thesis*
using *1 2* **by** *simp*
qed
moreover have *zip-cont t xs i* ∉ *?ys* **if** *i* > *0* **for** *i*
proof (*cases i* < *TT*)
case *True*
then have *dec* (*zip-cont t xs i*) =
(*map* (λ*j*. (*exec* (*t* + *fst* (*xs* ! *j*)) <:> *j*) *i*) [*0*..<*k*] @
map (λ*j*. *case snd* (*xs* ! *j*) *of None* ⇒ *0* | *Some d* ⇒ *if i* = *exec* (*t* + *d*) <#> *j then 1 else 0*) [*0*..<*k*] @
[*if i* < *t then 1 else 0*, *0*])
using *dec-zip-cont that* **by** *simp*
then have *dec* (*zip-cont t xs i*) ! (*2* ∗ *k* + *1*) = *0*
using *twok1-nth* **by** *force*
then show *?thesis*
by *simp*
next
case *False*
then have *zip-cont t xs i* = *0*
using *zip-cont-def* **by** *simp*
then show *?thesis*
by *simp*
qed
ultimately show *?thesis*
using *begin-tapeI* **by** *simp*
qed

definition *es6* ≡ *es5* @ *map* (λ*i*. (*1* + *Suc i*, *1* + *Suc i*)) [*0*..<*n*] @ [(*1* + *n*, *1* + *n*)]

lemma *len-es6*: *length es6* = *length* (*es-fmt n*) + *2* ∗ *TT* + *n* + *4*
using *es6-def len-es5* **by** *simp*

definition *tps6* :: *tape list* **where**
tps6 ≡
[(⌊*zs*⌋, *n* + *1*),
(*zip-cont 0* (*replicate k* (*0*, *Some 0*)), *n* + *1*)] @
replicate (*2* ∗ *k* + *1*) (⌈▷⌉)

lemma *tm6*: *traces tm6 tps0 es6 tps6*
unfolding *tm6-def es6-def*
proof (*rule traces-sequential*)
show *traces tm5 tps0 es5 tps5*
using *tm5* **.**
show *traces*
tm5-6
tps5
(*map* (λ*i*. (*1* + *Suc i*, *1* + *Suc i*)) [*0*..<*n*] @ [(*1* + *n*, *1* + *n*)])
tps6
unfolding *tm5-6-def*
proof (*rule traces-tm-trans-until-01I*[**where** *?n*=*n*])
show *1* < *length tps5*
using *tps5-def* **by** *simp*
show *rneigh* (*tps5* ! *0*) {*0*} *n*
using *tps5-def contents-def zs-proper* **by** (*intro rneighI*) *auto*

show *map* (*λi*. (*1* + *Suc i*, *1* + *Suc i*)) [*0*..<*n*] @ [(*1* + *n*, *1* + *n*)] =
map (*λi*. (*tps5* :#: *0* + *Suc i*, *tps5* :#: *1* + *Suc i*)) [*0*..<*n*] @ [(*tps5* :#: *0* + *n*, *tps5* :#: *1* + *n*)]
using *tps5-def* **by** *simp*
show *tps6* = *tps5*
[*0* := *tps5* ! *0* |+| *n*,
1 := *transplant* (*tps5* ! *0*) (*tps5* ! *1*) (*λh*. *enc* (*h mod G* # *replicate* (*k* − *1*) *0* @ *replicate k 0* @ [*0*, *0*]))
n]
proof −
define *f* **where** *f* = (*λh*. *enc* (*h mod G* # *replicate* (*k* − *1*) *0* @ *replicate k 0* @ [*0*, *0*]))
let *?tp1* = *tps5* ! *0*
let *?tp2* = *tps5* ! *1*
let *?xs* = *replicate k* (*0*::*nat*, *Some 0*::*nat option*)
have *rhs-less-TT*: *zip-cont 0 ?xs i* = *enc*
(*map* (*λj*. (*start-config k zs* <:> *j*) *i*) [*0*..<*k*] @
map (*λj*. *if i* = *start-config k zs* <#> *j then 1 else 0*) [*0*..<*k*] @
[*0*, *if i* = *0 then 1 else 0*])
if *i* < *TT* **for** *i*
proof −
have *zip-cont 0 ?xs i* = *enc*
(*map* (*λj*. (*exec* (*0* + *fst* (*?xs* ! *j*)) <:> *j*) *i*) [*0*..<*k*] @
map (*λj*. *case snd* (*?xs* ! *j*) *of None* ⇒ *0* | *Some d* ⇒ *if i* = *exec* (*0* + *d*) <#> *j then 1 else 0*)
[*0*..<*k*] @
[*if i* < *0 then 1 else 0*,
if i = *0 then 1 else 0*])
using *that zip-cont-def* **by** *simp*
moreover have *map* (*λj*. (*exec* (*0* + *fst* (*?xs* ! *j*)) <:> *j*) *i*) [*0*..<*k*] = *map* (*λj*. (*exec 0* <:> *j*) *i*) [*0*..<*k*]
by *simp*
ultimately have *zip-cont 0 ?xs i* = *enc*
(*map* (*λj*. (*exec 0* <:> *j*) *i*) [*0*..<*k*] @
map (*λj*. *case snd* (*?xs* ! *j*) *of None* ⇒ *0* | *Some d* ⇒ *if i* = *exec* (*0* + *d*) <#> *j then 1 else 0*)
[*0*..<*k*] @
[*if i* < *0 then 1 else 0*,
if i = *0 then 1 else 0*])
by *metis*
also have ... = *enc*
(*map* (*λj*. (*exec 0* <:> *j*) *i*) [*0*..<*k*] @
map (*λj*. *if i* = *exec 0* <#> *j then 1 else 0*) [*0*..<*k*] @
[*if i* < *0 then 1 else 0*,
if i = *0 then 1 else 0*])
proof −
have *1*: *map* (*λj*. *case snd* (*?xs* ! *j*) *of None* ⇒ *0* | *Some d* ⇒ *if i* = *exec* (*0* + *d*) <#> *j then 1 else 0*)
[*0*..<*k*] =
map (*λj*. *if i* = *exec 0* <#> *j then 1 else 0*) [*0*..<*k*]
by *simp*
show *?thesis*
by (*simp add*: *1*)
qed
finally show *?thesis*
using *exec-def* **by** *simp*
qed

have (*if snd ?tp2* ≤ *i* ∧ *i* < *snd ?tp2* + *n then f* (*fst ?tp1* (*snd ?tp1* + *i* − *snd ?tp2*)) *else fst ?tp2 i*) =
zip-cont 0 (*replicate k* (*0*, *Some 0*)) *i*
(**is** *?lhs* = *?rhs*)
for *i*
proof (*cases 1* ≤ *i* ∧ *i* < *1* + *n*)
case *True*
then have *snd ?tp2* ≤ *i* ∧ *i* < *snd ?tp2* + *n*
using *tps5-def* **by** *simp*
then have *?lhs* = *f* (*fst ?tp1* (*snd ?tp1* + *i* − *snd ?tp2*))
by *simp*
then have *?lhs* = *f* (*fst ?tp1 i*)
using *tps5-def* **by** *simp*

```
    then have ?lhs = f (zs ! (i - 1)) (is - = f (zs ! ?i))
      using tps5-def contents-def True by simp
    moreover have zs ! ?i < G
      using True zs-less-G by auto
    ultimately have lhs: ?lhs = enc (zs ! ?i # replicate (k - 1) 0 @ replicate k 0 @ [0, 0])
      using f-def by simp

    have TT > n
      using fmt-ge-n by (simp add: le-imp-less-Suc)
    then have i < TT
      using True by simp
    then have rhs: ?rhs = enc
        (map (λj. (start-config k zs <:> j) i) [0..<k] @
         map (λj. if i = start-config k zs <#> j then 1 else 0) [0..<k] @
         [0, 0])
      using True rhs-less-TT by simp

    have map (λj. (start-config k zs <:> j) i) [0..<k] = zs ! ?i # replicate (k - 1) 0 (is ?l = ?r)
    proof (intro nth-equalityI)
      show length ?l = length ?r
        using k-ge-2 by simp
      show ?l ! j = ?r ! j if j < length ?l for j
      proof (cases j = 0)
        case c1: True
        have (start-config k zs <:> 0) i = zs ! ?i
          using start-config-def True by simp
        then show ?thesis
          using c1 that by auto
      next
        case c2: False
        then have (start-config k zs <:> j) i = 0
          using start-config-def True that by simp
        then show ?thesis
          using c2 that by simp
      qed
    qed
    moreover have map (λj. if i = start-config k zs <#> j then 1 else 0) [0..<k] = replicate k 0
    proof -
      have start-config k zs <#> j = 0 if j < k for j
        using that start-config-pos by auto
      then have map (λj. if i = start-config k zs <#> j then 1 else 0) [0..<k] = map (λj. 0) [0..<k]
        using True by simp
      then show ?thesis
        by (simp add: map-replicate-trivial)
    qed
    ultimately show ?lhs = ?rhs
      using lhs rhs by (metis Cons-eq-appendI)
  next
    case outside: False
    show ?thesis
    proof (cases i = 0)
      case True
      then have lhs: ?lhs = enc (replicate k 1 @ replicate k 1 @ [0, 1])
        using tps5-def contents'-def by simp
      moreover have ?rhs = enc
          (map (λj. (start-config k zs <:> j) i) [0..<k] @
           map (λj. if i = start-config k zs <#> j then 1 else 0) [0..<k] @
           [0, 1])
        using rhs-less-TT True by simp
      moreover have map (λj. (start-config k zs <:> j) i) [0..<k] = replicate k 1 (is ?l = ?r)
      proof (rule nth-equalityI)
        show length ?l = length ?r
          by simp
```

then show ?l ! j = ?r ! j **if** j < length ?l **for** j
using start-config-def True that start-config2 **by** simp
qed
moreover have map (λj. if i = start-config k zs <#> j then 1 else 0) [0..<k] = replicate k 1 (**is** ?l = ?r)
proof (rule nth-equalityI)
show length ?l = length ?r
by simp
show ?l ! j = ?r ! j **if** j < length ?l **for** j
using True start-config-pos that **by** auto
qed
ultimately show ?thesis
by metis
next
case False
then have i > n
using outside **by** simp
then have lhs: ?lhs = fst ?tp2 i
using tps5-def **by** simp
then show ?thesis
proof (cases i < TT)
case True
moreover have i > 0
using False **by** simp
ultimately have lhs: ?lhs = enc (replicate k 0 @ replicate k 0 @ [0, 0])
using lhs tps5-def contents′-def **by** simp

have ?rhs = enc
(map (λj. (start-config k zs <:> j) i) [0..<k] @
map (λj. if i = start-config k zs <#> j then 1 else 0) [0..<k] @
[0, 0])
using True False rhs-less-TT **by** simp
moreover have map (λj. (start-config k zs <:> j) i) [0..<k] = replicate k 0 (**is** ?l = ?r)
proof (rule nth-equalityI)
show length ?l = length ?r
by simp
show ?l ! j = ?r ! j **if** j < length ?l **for** j
proof (cases j = 0)
case True
then show ?thesis
using that start-config-def ‹i > n› **by** simp
next
case False
then show ?thesis
using that start-config-def ‹i > 0› **by** simp
qed
qed
moreover have map (λj. if i = start-config k zs <#> j then 1 else 0) [0..<k] = replicate k 0
proof −
have start-config k zs <#> j = 0 **if** j < k **for** j
using that start-config-pos **by** auto
then have map (λj. if i = start-config k zs <#> j then 1 else 0) [0..<k] = map (λj. 0) [0..<k]
by (simp add: False)
then show ?thesis
by (simp add: map-replicate-trivial)
qed
ultimately show ?thesis
using lhs **by** metis
next
case False
then have i ≥ TT
by simp
moreover have length (enc (replicate k 1 @ replicate k 1 @ [0, 1]) # replicate (fmt n) (enc (replicate

k 0 @ replicate k 0 @ [0, 0]))) = TT
              by *simp*
            ultimately have ?lhs = 0
              using *lhs contents'-at-ge* **by** (*simp add*: *tps5-def*)
            moreover have ?rhs = 0
              using *zip-cont-def* ‹i ≥ TT› **by** *simp*
            ultimately show ?thesis
              by *simp*
          qed
        qed
      qed
      then have (λi. if snd ?tp2 ≤ i ∧ i < snd ?tp2 + n then f (fst ?tp1 (snd ?tp1 + i − snd ?tp2)) else fst ?tp2 i) =
            zip-cont 0 (replicate k (0, Some 0))
        by *simp*
      moreover have *transplant* (tps5 ! 0) (tps5 ! 1) (λh. enc (h mod G # replicate (k − 1) 0 @ replicate k 0 @ [0, 0])) n =
          (λi. if snd ?tp2 ≤ i ∧ i < snd ?tp2 + n then f (fst ?tp1 (snd ?tp1 + i − snd ?tp2)) else fst ?tp2 i,
            snd ?tp2 + n)
        using *transplant-def f-def* **by** *auto*
      ultimately have *transplant* (tps5 ! 0) (tps5 ! 1) (λh. enc (h mod G # replicate (k − 1) 0 @ replicate k 0 @ [0, 0])) n =
          (zip-cont 0 (replicate k (0, Some 0)), n + 1)
        using *tps5-def* **by** *simp*
      then have tps6 ! 1 = transplant (tps5 ! 0) (tps5 ! 1) (λh. enc (h mod G # replicate (k − 1) 0 @ replicate k 0 @ [0, 0])) n
        using *tps6-def* **by** *simp*
      moreover have tps6 ! 0 = tps5 ! 0 |+| n
        using *tps6-def tps5-def* **by** *simp*
      ultimately show ?thesis
        using *tps5-def tps6-def* **by** *simp*
    qed
  qed
qed

definition *tps7* :: *tape list* **where**
  tps7 ≡
    [(⌊zs⌋, n + 1),
     (zip-cont 0 (replicate k (0, Some 0)), 0)] @
    replicate (2 ∗ k + 1) (⌈▷⌉)

definition *es7* ≡ es6 @ map (λi. (n + 1, i)) (rev [0..<n + 1]) @ [(n + 1, 0)]

lemma *len-es7*: length es7 = length (es-fmt n) + 2 ∗ TT + 2 ∗ n + 6
  using *es7-def len-es6* **by** *simp*

lemma *tm7*: traces tm7 tps0 es7 tps7
  unfolding *tm7-def es7-def*
proof (*rule traces-sequential*)
  show traces tm6 tps0 es6 tps6
    using *tm6* **.**
  show traces tm-left-until1 tps6 (map (Pair (n + 1)) (rev [0..<n + 1]) @ [(n + 1, 0)]) tps7
  proof (*rule traces-tm-left-until-1I*)
    show 1 < length tps6
      using *tps6-def* **by** *simp*
    show map (Pair (n + 1)) (rev [0..<n + 1]) @ [(n + 1, 0)] =
        map (Pair (tps6 :#: 0)) (rev [0..<tps6 :#: 1]) @ [(tps6 :#: 0, 0)]
      using *tps6-def* **by** *simp*
    show tps7 = tps6[1 := tps6 ! 1 |#=| 0]
      using *tps6-def tps7-def* **by** *simp*
    show begin-tape StartSym (tps6 ! 1)
      using *begin-tape-zip-cont tps6-def* **by** *simp*
  qed

```
qed

definition tps8 :: tape list where
  tps8 ≡
    [(⌊zs⌋, n + 1),
     (zip-cont 0 (replicate k (0, Some 0)), 0),
     ⌈□⌉] @
    replicate (2 * k) (⌈▷⌉)

definition es8 ≡ es7 @ [(n + 1, 0)]

lemma len-es8: length es8 = length (es-fmt n) + 2 * TT + 2 * n + 7
  using es8-def len-es7 by simp

lemma tm8: traces tm8 tps0 es8 tps8
  unfolding tm8-def es8-def
proof (rule traces-sequential)
  show traces tm7 tps0 es7 tps7
    using tm7 .
  show traces (tm-write 2 0) tps7 [(n + 1, 0)] tps8
  proof (rule traces-tm-write-ge2I)
    show (2::nat) ≤ 2
      by simp
    show 2 < length tps7 [(n + 1, 0)] = [(tps7 :#: 0, tps7 :#: 1)]
      using tps7-def by simp-all
    show tps8 = tps7[2 := tps7 ! 2 |:=| 0]
    proof (rule nth-equalityI)
      show length tps8 = length (tps7[2 := tps7 ! 2 |:=| 0])
        using tps7-def tps8-def by simp
      show tps8 ! i = tps7[2 := tps7 ! 2 |:=| 0] ! i if i < length tps8 for i
      proof -
        consider i = 0 | i = 1 | i = 2 | i > 2
          by linarith
        then show ?thesis
        proof (cases)
          case 1
          then show ?thesis
            using tps7-def tps8-def by simp
        next
          case 2
          then show ?thesis
            using tps7-def tps8-def by simp
        next
          case 3
          then have *: tps8 ! i = ⌈□⌉
            using tps8-def by simp
          have (tps7 ! 2) |:=| □ = ⌈□⌉
            using tps7-def onesie-write by simp
          then have (tps7[2 := tps7 ! 2 |:=| □]) ! 2 = ⌈□⌉
            using tps7-def by simp
          then show ?thesis
            using 3 * by simp
        next
          case 4
          then have tps8 ! i = ⌈▷⌉
            using tps8-def that by simp
          moreover have tps7 ! i = ⌈▷⌉
            using tps7-def that 4 tps8-def by auto
          ultimately show ?thesis
            by (simp add: 4 less-not-refl3)
        qed
      qed
    qed
```

qed
qed

definition *tps9* :: *tape list* **where**
tps9 ≡
[(⌊*zs*⌋, *n* + *1*),
(*zip-cont 0* (*replicate k* (*0*, *Some 0*)), *0*),
⌈□⌉] @
replicate (*2* ∗ *k*) (⌈□⌉)

definition *es9* ≡ *es8* @ [(*n* + *1*, *0*)]

lemma *len-es9*: *length es9* = *length* (*es-fmt n*) + *2* ∗ *TT* + *2* ∗ *n* + *8*
using *es9-def len-es8* **by** *simp*

lemma *tm9*: *traces tm9 tps0 es9 tps9*
unfolding *tm9-def es9-def*
proof (*rule traces-sequential*[*OF tm8*])
show *traces* (*tm-write-many* {*3*..<*2* ∗ *k* + *3*} *0*) *tps8* [(*n* + *1*, *0*)] *tps9*
proof (*rule traces-tm-write-manyI*[**where** *?k*=*2*∗*k*+*3*])
show *0* ∉ {*3*..<*2* ∗ *k* + *3*}
by *simp*
show ∀ *j*∈ {*3*..<*2* ∗ *k* + *3*}. *j* < *2* ∗ *k* + *3*
by *simp*
show *2* ≤ *2* ∗ *k* + *3*
by *simp*
show *length tps8* = *2* ∗ *k* + *3* *length tps9* = *2* ∗ *k* + *3*
using *tps8-def tps9-def* **by** *simp-all*
show [(*n* + *1*, *0*)] = [(*tps8* :#: *0*, *tps8* :#: *1*)]
using *tps8-def* **by** *simp*
show *tps9* ! *j* = *tps8* ! *j* **if** *j* < *2* ∗ *k* + *3* *j* ∉ {*3*..<*2* ∗ *k* + *3*} **for** *j*
proof −
have *j* < *3*
using *that* **by** *simp*
then show *?thesis*
using *tps8-def tps9-def* **by** (*metis* (*no-types*, *lifting*) *length-Cons list.size*(*3*) *nth-append numeral-3-eq-3*)
qed
show *tps9* ! *j* = *tps8* ! *j* |:=| *0* **if** *j* ∈ {*3*..<*2* ∗ *k* + *3*} **for** *j*
proof −
have *j*: *j* ≥ *3* *j* < *2* ∗ *k* + *3*
using *that* **by** *simp-all*
then have *tps8* ! *j* = ⌈▷⌉
using *tps8-def* **by** *simp*
moreover have *tps9* ! *j* = ⌈□⌉
using *j tps9-def* **by** *simp*
ultimately show *?thesis*
by (*simp add*: *onesie-write*)
qed
qed
qed

The loop

Immediately before and during the loop the tapes will have the shape below. The input tape will stay unchanged. The output tape will contain the k encoded tapes of M. The first memorization tape will contain M's state. The following k memorization tapes will store information about head movements. The final k memorization tapes will have information about read or to-be-written symbols.

definition *tpsL* :: *nat* ⇒ (*nat* × *nat option*) *list* ⇒ *nat* ⇒ *nat* ⇒ (*nat* ⇒ *nat*) ⇒ (*nat* ⇒ *symbol*) ⇒ *tape list*
where
tpsL t xs i q mvs symbs ≡
(⌊*zs*⌋, *n* + *1*) #
(*zip-cont t xs*, *i*) #
⌈*fst* (*exec* (*t* + *q*))⌉ #

map (*onesie* ∘ *mvs*) [*0*..<*k*] @
map (*onesie* ∘ *symbs*) [*0*..<*k*]

lemma *length-tpsL* [*simp*]: *length* (*tpsL t xs i q mvs symbs*) = *2* ∗ *k* + *3*
using *tpsL-def* **by** *simp*

lemma *tpsL-pos-0*: *tpsL t xs i q mvs symbs* :#: *0* = *n* + *1*
unfolding *tpsL-def* **by** *simp*

lemma *tpsL-pos-1*: *tpsL t xs i q mvs symbs* :#: *1* = *i*
unfolding *tpsL-def* **by** *simp*

lemma *read-tpsL-0*: *read* (*tpsL t xs i q mvs symbs*) ! *0* = □
unfolding *tpsL-def* **using** *contents-def read-def* **by** *simp*

lemma *read-tpsL-1*: *read* (*tpsL t xs i q mvs symbs*) ! *1* =
(*if i* < *TT then enc*
(*map* (λ*j*. (*exec* (*t* + *fst* (*xs* ! *j*)) <:> *j*) *i*) [*0*..<*k*] @
map (λ*j*. *case snd* (*xs* ! *j*) *of None* ⇒ *0* | *Some d* ⇒ *if i* = *exec* (*t* + *d*) <#> *j then 1 else 0*) [*0*..<*k*] @
[*if i* < *t then 1 else 0*,
if i = *0 then 1 else 0*])
else 0)
unfolding *tpsL-def zip-cont-def* **using** *read-def* **by** *simp*

lemma *read-tpsL-1-nth-less-k*:
assumes *i* < *TT* **and** *j* < *k*
shows *enc-nth* (*read* (*tpsL t xs i q mvs symbs*) ! *1*) *j* = (*exec* (*t* + *fst* (*xs* ! *j*)) <:> *j*) *i*
using *assms read-tpsL-1 dec-zip-cont-less-k enc-nth-def zip-cont-def* **by** *auto*

lemma *read-tpsL-1-nth-less-2k*:
assumes *i* < *TT* **and** *k* ≤ *j* **and** *j* < *2* ∗ *k*
shows *enc-nth* (*read* (*tpsL t xs i q mvs symbs*) ! *1*) *j* =
(*case snd* (*xs* ! (*j* − *k*)) *of None* ⇒ *0* | *Some d* ⇒ *if i* = *exec* (*t* + *d*) <#> (*j* − *k*) *then 1 else 0*)
using *assms read-tpsL-1 dec-zip-cont-less-2k enc-nth-def zip-cont-def* **by** *simp*

lemma *read-tpsL-1-nth-2k*:
assumes *i* < *TT*
shows *enc-nth* (*read* (*tpsL t xs i q mvs symbs*) ! *1*) (*2* ∗ *k*) = (*if i* < *t then 1 else 0*)
using *assms read-tpsL-1 dec-zip-cont-2k enc-nth-def zip-cont-def* **by** *simp*

lemma *read-tpsL-1-nth-2k1*:
assumes *i* < *TT*
shows *enc-nth* (*read* (*tpsL t xs i q mvs symbs*) ! *1*) (*2* ∗ *k* + *1*) = (*if i* = *0 then 1 else 0*)
using *assms read-tpsL-1 dec-zip-cont-2k1 enc-nth-def zip-cont-def* **by** *simp*

lemma *read-tpsL-1-bounds*:
assumes *i* < *TT*
shows *read* (*tpsL t xs i q mvs symbs*) ! *1* > *1*
and *read* (*tpsL t xs i q mvs symbs*) ! *1* < *G* ^ (*2* ∗ *k* + *2*) + *2*
proof −
have *read* (*tpsL t xs i q mvs symbs*) ! *1* = *zip-cont t xs i*
using *tpsL-def read-def* **by** *simp*
then show *read* (*tpsL t xs i q mvs symbs*) ! *1* > *1*
and *read* (*tpsL t xs i q mvs symbs*) ! *1* < *G* ^ (*2* ∗ *k* + *2*) + *2*
using *zip-cont-gt-1*[*OF assms*] *zip-cont-less*[*OF assms*] **by** *simp-all*
qed

lemma *read-tpsL-2*: *read* (*tpsL t xs i q mvs symbs*) ! *2* = *fst* (*exec* (*t* + *q*))
unfolding *tpsL-def* **using** *contents-def read-def* **by** *simp*

lemma *read-tpsL-3*:
assumes *3* ≤ *j* **and** *j* < *k* + *3*
shows *read* (*tpsL t xs i q mvs symbs*) ! *j* = *mvs* (*j* − *3*)

proof −
  define j' **where** $j' = j - 3$
  then have $j' < k$
    using *assms* **by** *simp*
  have *read* (*tpsL t xs i q mvs symbs*) ! j =
      (*map* (*tape-read* ∘ (*onesie* ∘ *mvs*)) [*0*..<*k*] @
       map (*tape-read* ∘ (*onesie* ∘ *symbs*)) [*0*..<*k*]) !
      (j − *Suc* (*Suc* (*Suc* *0*)))
    unfolding *tpsL-def read-def* **using** *assms* **by** *simp*
  also have ... =
    (*map* (*tape-read* ∘ (*onesie* ∘ *mvs*)) [*0*..<*k*] @
     map (*tape-read* ∘ (*onesie* ∘ *symbs*)) [*0*..<*k*]) ! j'
    by (*simp add*: j'*-def numeral-3-eq-3*)
  also have ... = *mvs* j'
    using ‹$j' < k$› **by** (*simp add*: *nth-append*)
  finally have *read* (*tpsL t xs i q mvs symbs*) ! j = *mvs* j' **.**
  then show *?thesis*
    using j'*-def* **by** *simp*
qed

lemma *read-tpsL-3'*:
  assumes $j < k$
  shows *read* (*tpsL t xs i q mvs symbs*) ! ($j + 3$) = *mvs j*
  using *assms read-tpsL-3* **by** *simp*

lemma *read-tpsL-4*:
  assumes $k + 3 \leq j$ **and** $j < 2 * k + 3$
  shows *read* (*tpsL t xs i q mvs symbs*) ! j = *symbs* ($j - k - 3$)
proof −
  define j' **where** $j' = j - 3$
  then have j': $k \leq j'$ $j' < k + k$
    using *assms* **by** *simp-all*
  have *read* (*tpsL t xs i q mvs symbs*) ! j =
      (*map* (*tape-read* ∘ (*onesie* ∘ *mvs*)) [*0*..<*k*] @
       map (*tape-read* ∘ (*onesie* ∘ *symbs*)) [*0*..<*k*]) !
      (j − *Suc* (*Suc* (*Suc* *0*)))
    unfolding *tpsL-def read-def* **using** *assms* **by** *simp*
  also have ... =
    (*map* (*tape-read* ∘ (*onesie* ∘ *mvs*)) [*0*..<*k*] @
     map (*tape-read* ∘ (*onesie* ∘ *symbs*)) [*0*..<*k*]) ! j'
    by (*simp add*: j'*-def numeral-3-eq-3*)
  also have ... = *symbs* ($j' - k$)
    using j' **by** (*simp add*: *nth-append*)
  finally have *read* (*tpsL t xs i q mvs symbs*) ! j = *symbs* ($j' - k$) **.**
  then show *?thesis*
    using j'*-def* **using** *diff-commute* **by** *auto*
qed

lemma *map-const-upt*: *map* (*onesie* ∘ (λ-. *c*)) [*0*..<*k*] = *replicate k* ⌈*c*⌉
  by (*metis List.map.compositionality map-replicate map-replicate-trivial*)

After the initialization, that is, right before the loop, the simulator's tapes look like this:

lemma *tps9-tpsL*: *tps9* = *tpsL 0* (*replicate k* (*0*, *Some 0*)) *0 0* (λ*j*. *0*) (λ*j*. *0*)
proof −
  have *fst* (*exec 0*) = *0*
    using *exec-def* **by** (*simp add*: *start-config-def*)
  then have *tpsL 0* (*replicate k* (*0*, *Some 0*)) *0 0* (λ*j*. *0*) (λ*j*. *0*) =
    (⌊*zs*⌋, $n + 1$) #
    (*zip-cont 0* (*replicate k* (*0*, *Some 0*)), *0*) #
    ⌈□⌉ #
    replicate k (⌈□⌉) @
    replicate k (⌈□⌉)
    using *tpsL-def map-const-upt* **by** *simp*

then show *?thesis*
using *tps9-def* **by** (*metis Cons-eq-appendI mult-2 replicate-add self-append-conv2*)
qed

lemma *tpsL-0*: *tpsL t xs i q mvs symbs* ! *0* = ($\lfloor zs \rfloor$, *n* + *1*)
using *tpsL-def* **by** *simp*

lemma *tpsL-1*: *tpsL t xs i q mvs symbs* ! *1* = (*zip-cont t xs*, *i*)
using *tpsL-def* **by** *simp*

lemma *tpsL-2*: *tpsL t xs i q mvs symbs* ! *2* = $\lceil$*fst* (*exec* (*t* + *q*))$\rceil$
using *tpsL-def* **by** *simp*

lemma *tpsL-mvs*: *j* < *k* $\Longrightarrow$ *tpsL t xs i q mvs symbs* ! (*3* + *j*) = $\lceil$*mvs j*$\rceil$
using *tpsL-def* **by** (*simp add*: *nth-append*)

lemma *tpsL-mvs'*: *3* ≤ *j* $\Longrightarrow$ *j* < *3* + *k* $\Longrightarrow$ *tpsL t xs i q mvs symbs* ! *j* = $\lceil$*mvs* (*j* − *3*)$\rceil$
using *tpsL-mvs* **by** (*metis add.commute le-add-diff-inverse less-diff-conv2*)

lemma *tpsL-symbs*:
assumes *j* < *k*
shows *tpsL t xs i q mvs symbs* ! (*3* + *k* + *j*) = $\lceil$*symbs j*$\rceil$
using *tpsL-def assms* **by** (*simp add*: *nth-append*)

lemma *tpsL-symbs'*:
assumes *3* + *k* ≤ *j* **and** *j* < *2* ∗ *k* + *3*
shows *tpsL t xs i q mvs symbs* ! *j* = $\lceil$*symbs* (*j* − *k* − *3*)$\rceil$
proof −
have *j* − (*k* + *3*) < *k*
using *assms*(*1*) *assms*(*2*) **by** *linarith*
then show *?thesis*
using *assms*(*1*) *tpsL-symbs* **by** *fastforce*
qed

The condition: less than *TT* steps simulated.

definition *tpsC0* :: *nat* ⇒ *tape list* **where**
tpsC0 t ≡ *tpsL t* (*replicate k* (*0*, *Some 0*)) *0 0* (λ*j*. *0*) (λ*j*. *0*)

definition *tpsC1 t* ≡ *tpsL t* (*replicate k* (*0*, *Some 0*)) *t 0* (λ*j*. *0*) (λ*j*. *0*)

definition *esC t* ≡ *map* (λ*i*. (*n* + *1*, *Suc i*)) [*0*..<*t*] @ [(*n* + *1*, *t*)]

lemma *set-filter-upt*: *x* ∈ *set* (*filter f* [*0*..<*N*]) $\Longrightarrow$ *x* < *N*
by *simp*

lemma *set-filter-upt'*: *x* ∈ *set* (*filter f* [*0*..<*N*]) $\Longrightarrow$ *f x*
by *simp*

We will use this TM at the end of the loop again. Therefore we prove a more general result than necessary at this point.

lemma *tmC-general*:
assumes *t* ≤ *TT*
and *tps* = *tpsL t xs 0 i mvs symbs*
and *tps'* = *tpsL t xs t i mvs symbs*
shows *traces tmC tps* (*esC t*) *tps'*
unfolding *tmC-def*
proof (*rule traces-tm-right-until-1I*[**where** *?n*=*t*])
show *1* < *length tps*
using *assms*(*2*) **by** *simp*
show *tps'* = *tps*[*1* := *tps* ! *1* |+| *t*]
using *assms*(*2*,*3*) *tpsL-def* **by** *simp*
show *esC t* =
map (λ*i*. (*tps* :#: *0*, *tps* :#: *1* + *Suc i*)) [*0*..<*t*] @ [(*tps* :#: *0*, *tps* :#: *1* + *t*)]

```
    using esC-def assms(2) tpsL-def by simp
  show rneigh (tps ! 1) {y. y < G ^ (2 ∗ k + 2) + 2 ∧ (y = 0 ∨ dec y ! (2 ∗ k) = 0)} t
    (is rneigh - ?s t)
  proof (rule rneighI)
    have 1: tps ! 1 = (zip-cont t xs, 0)
      using assms(2) tpsL-def by simp
    have s: ?s = {y. y = 0 ∨ (dec y ! (2 ∗ k) = 0 ∧ y < G ^ (2 ∗ k + 2) + 2)} (is - = ?r)
      by auto
    show (tps ::: 1) (tps :#: 1 + t) ∈ ?s
    proof (cases t = TT)
      case True
      then have tps :#: 1 + t = TT
        using assms(2) tpsL-def by simp
      moreover have (tps ::: 1) TT = 0
        using assms(2) tpsL-def zip-cont-def by simp
      ultimately show ?thesis
        by auto
    next
      case False
      with assms have t < TT
        by simp
      let ?y = (tps ::: 1) t
      have dec ?y ! (2 ∗ k) = 0
        using assms(2) tpsL-def dec-zip-cont[OF ‹t < TT›] by (simp add: twok-nth)
      moreover have ?y < G ^ (2 ∗ k + 2) + 2
        using assms(2) tpsL-def ‹t < TT› zip-cont-less by simp
      ultimately have ?y ∈ ?s
        using s by simp
      then show ?thesis
        using 1 by simp
    qed
    show (tps ::: 1) (tps :#: 1 + m) ∉ ?s (is ?y ∉ ?s) if m < t for m
    proof −
      have m < TT
        using that assms by simp
      then have dec ?y ! (2 ∗ k) = 1
        using tpsC0-def tpsL-def dec-zip-cont[OF ‹m < TT›] that
        by (metis (no-types, lifting) 1 add-cancel-right-left dec-zip-cont-2k prod.sel(1) prod.sel(2))
      moreover from ‹m < TT› have ?y > 1
        using 1 zip-cont-gt-1 by simp
      ultimately show ?thesis
        using s by simp
    qed
  qed
qed

corollary tmC:
  assumes t ≤ TT
  shows traces tmC (tpsC0 t) (esC t) (tpsC1 t)
  using tmC-general tpsC0-def tpsC1-def assms by simp

lemma tpsC1-at-T: tpsC1 TT :.: 1 = 0
  using tpsC1-def tpsL-def zip-cont-def by simp

lemma tpsC1-at-less-T: t < TT ⟹ tpsC1 t :.: 1 > 0
proof −
  assume t < TT
  then have tpsC1 t :.: 1 > 1
    using tpsC1-def tpsL-def zip-cont-def enc-greater by simp
  then show ?thesis
    by simp
qed
```

The body of the loop: simulating one step

```
definition tpsL0 t ≡ tpsL t (replicate k (0, Some 0)) t 0 (λj. 0) (λj. 0)

lemma tpsL0-eq-tpsC1: tpsL0 t = tpsC1 t
  using tpsL0-def tpsC1-def by simp

definition esL1 t ≡ map (λi. (n + 1, i)) (rev [0..<t]) @ [(n + 1, 0)]

definition tpsL1 t ≡ tpsL t (replicate k (0, Some 0)) 0 0 (λj. 0) (λj. 0)

lemma tmL1: traces tmL1 (tpsL0 t) (esL1 t) (tpsL1 t)
  unfolding tmL1-def esL1-def
proof (rule traces-tm-left-until-1I)
  show 1 < length (tpsL0 t)
    using tpsL0-def tpsL-def by simp
  show map (Pair (n + 1)) (rev [0..<t]) @ [(n + 1, 0)] =
      map (Pair (tpsL0 t :#: 0)) (rev [0..<tpsL0 t :#: 1]) @ [(tpsL0 t :#: 0, 0)]
    using tpsL0-def tpsL-def by simp
  show tpsL1 t = (tpsL0 t)[1 := tpsL0 t ! 1 |#=| 0]
    using tpsL0-def tpsL1-def tpsL-def by simp
  show begin-tape StartSym (tpsL0 t ! 1)
    using begin-tape-zip-cont tpsL0-def tpsL-def by simp
qed
```

Collecting the read symbols of the simulated machines:

```
lemma sem-cmdL2-ge-TT:
  assumes tps = tpsL t xs i q mvs symbs and i ≥ TT
  shows sem cmdL2 (0, tps) = (1, tps)
proof (rule semI[of 2 * k + 3])
  show proper-command (2 * k + 3) cmdL2
    using cmdL2-def by simp
  show len: length tps = 2 * k + 3
    using assms(1) by simp
  show length tps = 2 * k + 3
    using assms(1) by simp
  let ?rs = read tps
  show fst (cmdL2 ?rs) = 1
  proof -
    have ?rs ! 1 = □
      using assms read-tpsL-1 by auto
    then show ?thesis
      by (simp add: cmdL2-def)
  qed
  then show act (cmdL2 ?rs [!] i) (tps ! i) = tps ! i if i < 2 * k + 3 for i
    using assms that
    by (metis (no-types, lifting) One-nat-def act-Stay cmdL2-at-eq-0 cmdL2-def len less-Suc-eq-0-disj
      less-numeral-extra(4) prod.sel(1) read-length)
qed

lemma sem-cmdL2-less-TT:
  assumes tps = tpsL t xs i q mvs symbs
    and symbs = (λj. if exec t <#> j < i then exec t <.> j else 0)
    and tps' = tpsL t xs (Suc i) q mvs symbs'
    and symbs' = (λj. if exec t <#> j < Suc i then exec t <.> j else 0)
    and i < TT
    and xs = replicate k (0, Some 0)
  shows sem cmdL2 (0, tps) = (0, tps')
proof (rule semI[of 2 * k + 3])
  show proper-command (2 * k + 3) cmdL2
    using cmdL2-def by simp
  show len: length tps = 2 * k + 3
    using assms(1) by simp
  show len': length tps' = 2 * k + 3
    using assms(3) by simp
```

define *rs* **where** $rs = read\ tps$
then have *rs-at-1*: $rs\ !\ 1 \neq \Box$
using *assms*(*1*,*5*) *read-tpsL-1 enc-greater* **by** (*metis* (*no-types*, *lifting*) *not-one-less-zero*)
then show $fst\ (cmdL2\ (read\ tps)) = 0$
by (*simp add*: *cmdL2-def rs-def*)
show $act\ (cmdL2\ (read\ tps)\ [!]\ j)\ (tps\ !\ j) = tps'\ !\ j$ **if** $j < 2 * k + 3$ **for** *j*
proof −
have *snd*: $snd\ (cmdL2\ rs) =$
$[(rs!0,\ Stay),\ (rs!1,\ Right),\ (rs!2,\ Stay)]$ @
(*map* ($\lambda j.\ (rs\ !\ (j + 3),\ Stay)$) $[0..<k]$) @
(*map* ($\lambda j.$ (*if* $1 < rs\ !\ 1 \wedge rs\ !\ 1 < G\hat{\ }(2*k+2)+2 \wedge$ *enc-nth* $(rs!1)\ (k+j) = 1$ *then enc-nth* $(rs!1)$ *j else* $rs!(3+k+j)$, *Stay*)) $[0..<k]$)
using *rs-at-1* **by** (*simp add*: *cmdL2-def*)
consider $j = 0 \mid j = 1 \mid j = 2 \mid 3 \leq j \wedge j < k + 3 \mid k + 3 \leq j \wedge j < 2 * k + 3$
using ‹$j < 2 * k + 3$› **by** *linarith*
then show *?thesis*
proof *cases*
case *1*
then have $cmdL2\ rs\ [!]\ j = (rs\ !\ 0,\ Stay)$
by (*simp add*: *snd*)
then show *?thesis*
using *act-Stay assms*(*1*,*3*) *tpsL-def 1 rs-def read-tpsL-0* **by** *auto*
next
case *2*
then have $cmdL2\ rs\ [!]\ j = (rs\ !\ 1,\ Right)$
by (*simp add*: *snd*)
then show *?thesis*
using *act-Right assms*(*1*,*3*) *2 rs-def*
by (*metis Suc-eq-plus1 add.commute add-Suc fst-conv len less-add-Suc1 numeral-3-eq-3 sndI tpsL-1*)
next
case *3*
then have $cmdL2\ rs\ [!]\ j = (rs\ !\ 2,\ Stay)$
by (*simp add*: *snd*)
then show *?thesis*
using *act-Stay assms*(*1*,*3*) *3 rs-def* **by** (*metis len that tpsL-2*)
next
case *4*
then have $cmdL2\ rs\ [!]\ j = (rs\ !\ j,\ Stay)$
using *cmdL2-at-3 rs-at-1* **by** *simp*
then show *?thesis*
using *act-Stay assms*(*1*,*3*) *4 rs-def* **by** (*metis add.commute len that tpsL-mvs'*)
next
case *5*
then have *1*: $cmdL2\ rs\ [!]\ j =$
(*if* $1 < rs\ !\ 1 \wedge rs\ !\ 1 < G\hat{\ }(2*k+2)+2 \wedge$ *enc-nth* $(rs\ !\ 1)\ (j - 3) = 1$
then enc-nth $(rs\ !\ 1)\ (j - k - 3)$
else $rs\ !\ j$,
Stay)
using *cmdL2-at-4 rs-at-1* **by** *simp*
have *enc*: $rs\ !\ 1 = enc$
(*map* ($\lambda j.$ (*exec* $(t + fst\ (xs\ !\ j))$ <:> *j*) *i*) $[0..<k]$ @
map ($\lambda j.$ *case snd* $(xs\ !\ j)$ *of None* $\Rightarrow 0 \mid$ *Some d* $\Rightarrow$ *if* $i =$ *exec* $(t + d)$ <#> *j then 1 else 0*) $[0..<k]$
@
[*if* $i < t$ *then 1 else 0*,
if $i = 0$ *then 1 else 0*])
using *read-tpsL-1 assms*(*1*,*5*) *rs-def* **by** *simp*
have $rs\ !\ 1 > 1\ rs\ !\ 1 < G\ \hat{\ }(2 * k + 2) + 2$
using *rs-def assms*(*1*,*5*) *read-tpsL-1-bounds* **by** *simp-all*
then have *cmd1*: $cmdL2\ rs\ [!]\ j =$
(*if enc-nth* $(rs\ !\ 1)\ (j - 3) = 1$ *then enc-nth* $(rs\ !\ 1)\ (j - k - 3)$ *else* $rs\ !\ j$, *Stay*)
using *1* **by** *simp*
have *enc-nth* $(rs\ !\ 1)\ (j - 3) =$
(*case snd* $(xs\ !\ (j - 3 - k))$ *of None* $\Rightarrow 0 \mid$ *Some d* $\Rightarrow$ *if* $i =$ *exec* $(t + d)$ <#> $(j - 3 - k)$ *then 1 else*

0)
    using *read-tpsL-1-nth-less-2k*[**where** *?j=j − 3*] *5 assms*(*1*,*5*) *rs-def* **by** *auto*
  then have *enc-nth* (*rs* ! *1*) (*j* − *3*) = (*if i* = *exec t* <#> (*j* − *3* − *k*) *then 1 else 0*)
    using *5 assms*(*6*) **by** *auto*
  then have *cmd2*: *enc-nth* (*rs* ! *1*) (*j* − *3*) = (*if i* = *exec t* <#> (*j* − *k* − *3*) *then 1 else 0*)
    by (*metis diff-right-commute*)
  let *?j* = *j* − *k* − *3*
  have *enc-nth* (*rs* ! *1*) *?j* = (*exec* (*t* + *fst* (*xs* ! *?j*)) <:> *?j*) *i*
    using *read-tpsL-1-nth-less-k*[**where** *?j=j − k − 3*] *5 assms*(*1*,*5*) *rs-def* **by** *auto*
  then have *enc-nth* (*rs* ! *1*) *?j* = (*exec t* <:> *?j*) *i*
    using *assms*(*6*) *5* **by** *auto*
  then have *cmdL2 rs* [!] *j* =
      (*if i* = *exec t* <#> *?j then* (*exec t* <:> *?j*) *i else rs* ! *j*, *Stay*)
    using *cmd1 cmd2* **by** *simp*
  then have *command*: *cmdL2 rs* [!] *j* =
      (*if i* = *exec t* <#> *?j then exec t* <.> *?j else rs* ! *j*, *Stay*)
    by *simp*

  have *tps*: *tps* ! *j* = ⌈*if exec t* <#> *?j* < *i then exec t* <.> *?j else* □⌉
    using *5 assms*(*1*,*2*) *tpsL-symbs′* **by** *simp*

  have *tps′*: *tps′* ! *j* = ⌈*if exec t* <#> *?j* < *Suc i then exec t* <.> *?j else* □⌉
    using *5 assms*(*3*,*4*) *tpsL-symbs′* **by** *simp*

  show *act* (*cmdL2* (*read tps*) [!] *j*) (*tps* ! *j*) = *tps′* ! *j*
  proof (*cases exec t* <#> *?j* = *i*)
    case *True*
    then have *act* (*cmdL2* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*exec t* <.> *?j*, *Stay*) (*tps* ! *j*)
      using *rs-def command* **by** *simp*
    also have ... = *act* (*exec t* <.> *?j*, *Stay*) ⌈*if exec t* <#> *?j* < *i then exec t* <.> *?j else* □⌉
      using *tps* **by** *simp*
    also have ... = ⌈*exec t* <.> *?j*⌉
      using *act-onesie* **by** *simp*
    also have ... = ⌈*if exec t* <#> *?j* < *Suc i then exec t* <.> *?j else* □⌉
      using *True* **by** *simp*
    also have ... = *tps′* ! *j*
      using *tps′* **by** *simp*
    finally show *?thesis* **.**
  next
    case *False*
    then have *act* (*cmdL2* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*rs* ! *j*, *Stay*) (*tps* ! *j*)
      using *rs-def command* **by** *simp*
    also have ... = *tps* ! *j*
      using *rs-def act-Stay* **by** (*simp add*: *5 len*)
    also have ... = ⌈*if exec t* <#> *?j* < *i then exec t* <.> *?j else* □⌉
      using *tps* **by** *simp*
    also have ... = ⌈*if exec t* <#> *?j* < *Suc i then exec t* <.> *?j else* □⌉
      using *False* **by** *simp*
    also have ... = *tps′* ! *j*
      using *tps′* **by** *simp*
    finally show *?thesis* **.**
  qed
 qed
qed
qed

corollary *sem-cmdL2-less-Tfmt*:
  assumes *xs* = *replicate k* (*0*, *Some 0*) **and** *i* < *TT*
  shows *sem cmdL2*
    (*0*, *tpsL t xs i q mvs* (λ*j*. *if exec t* <#> *j* < *i then exec t* <.> *j else* □)) =
    (*0*, *tpsL t xs* (*Suc i*) *q mvs* (λ*j*. *if exec t* <#> *j* < *Suc i then exec t* <.> *j else* □))
  using *sem-cmdL2-less-TT assms* **by** *simp*

```
lemma execute-cmdL2-le-TT:
  assumes tt ≤ TT and xs = replicate k (0, Some 0) and tps = tpsL t xs 0 q mvs (λ-. □)
  shows execute tmL1-2 (0, tps) tt =
    (0, tpsL t xs tt q mvs (λj. if exec t <#> j < tt then exec t <.> j else □))
  using assms(1)
proof (induction tt)
  case 0
  then show ?case
    using assms(3) by simp
next
  case (Suc tt)
  then have execute tmL1-2 (0, tps) (Suc tt) = exe tmL1-2 (execute tmL1-2 (0, tps) tt)
    by simp
  also have ... = exe tmL1-2 (0, tpsL t xs tt q mvs (λj. if exec t <#> j < tt then exec t <.> j else □))
    using Suc by simp
  also have ... = sem cmdL2 (0, tpsL t xs tt q mvs (λj. if exec t <#> j < tt then exec t <.> j else □))
    unfolding tmL1-2-def using Suc by (simp add: exe-lt-length)
  also have ... = (0, tpsL t xs (Suc tt) q mvs (λj. if exec t <#> j < Suc tt then exec t <.> j else □))
    using sem-cmdL2-less-Tfmt[OF assms(2)] Suc by simp
  finally show ?case .
qed

lemma tpsL-symbs-eq:
  assumes ⋀j. j < k ⟹ symbs j = symbs' j
  shows tpsL t xs i q mvs symbs = tpsL t xs i q mvs symbs'
  unfolding tpsL-def using assms by simp

lemma execute-cmdL2-Suc-TT:
  assumes xs = replicate k (0, Some 0) and tps = tpsL t xs 0 q mvs (λj. 0) and t < TT
  shows execute tmL1-2 (0, tps) (Suc TT) = (1, tpsL t xs TT q mvs (λj. exec t <.> j))
proof –
  have execute tmL1-2 (0, tps) (Suc TT) = exe tmL1-2 (execute tmL1-2 (0, tps) TT)
    by simp
  also have ... = exe tmL1-2 (0, tpsL t xs TT q mvs (λj. if exec t <#> j < TT then exec t <.> j else □))
    using execute-cmdL2-le-TT[where ?tt=TT] assms(1,2) by simp
  also have ... = sem cmdL2 (0, tpsL t xs TT q mvs (λj. if exec t <#> j < TT then exec t <.> j else □))
    unfolding tmL1-2-def by (simp add: exe-lt-length)
  also have ... = (1, tpsL t xs TT q mvs (λj. if exec t <#> j < TT then exec t <.> j else □))
    using sem-cmdL2-ge-TT[where ?i=TT] by simp
  finally have execute tmL1-2 (0, tps) (Suc TT) =
    (1, tpsL t xs TT q mvs (λj. if exec t <#> j < TT then exec t <.> j else □)) .
  moreover have (λj. if exec t <#> j < TT then exec t <.> j else □) j = (λj. exec t <.> j) j
      if j < k for j
    using exec-pos-less-TT assms(3) that by simp
  ultimately show ?thesis
    using tpsL-symbs-eq by fastforce
qed

definition esL1-2 ≡ map (λi. (n + 1, Suc i)) [0..<TT] @ [(n + 1, TT)]

lemma traces-tmL1-2:
  assumes xs = replicate k (0, Some 0) and t < TT
  shows traces tmL1-2 (tpsL t xs 0 q mvs (λ-. □)) esL1-2 (tpsL t xs TT q mvs (λj. exec t <.> j))
proof
  have len: length esL1-2 = Suc TT
    unfolding esL1-2-def by simp
  let ?tps = tpsL t xs 0 q mvs (λj. 0)
  show execute tmL1-2 (0, ?tps) (length esL1-2) =
      (length tmL1-2, tpsL t xs (Suc (fmt n)) q mvs (λj. exec t <.> j))
    using len execute-cmdL2-Suc-TT[OF assms(1) - assms(2)] by (simp add: tmL1-2-def)
  show ⋀i. i < length esL1-2 ⟹
        fst (execute tmL1-2 (0, ?tps) i) < length tmL1-2
    using len tmL1-2-def execute-cmdL2-le-TT assms(1)
```

by (*metis* (*no-types*, *lifting*) *One-nat-def Pair-inject length-Cons less-Suc-eq-le less-one list.size*(*3*) *prod.collapse*)
show *snd* (*execute tmL1-2* (*0*, *?tps*) (*Suc i*)) :#: *0* = *fst* (*esL1-2* ! *i*) ∧
snd (*execute tmL1-2* (*0*, *?tps*) (*Suc i*)) :#: *1* = *snd* (*esL1-2* ! *i*)
if *i* < *length esL1-2* **for** *i*
proof (*cases i* < *TT*)
case *True*
then have *execute tmL1-2* (*0*, *?tps*) (*Suc i*) =
(*0*, *tpsL t xs* (*Suc i*) *q mvs* (λ*j*. *if exec t* <#> *j* < *Suc i then exec t* <.> *j else* □))
using *execute-cmdL2-le-TT*[*of Suc i xs*] *assms* **by** *simp*
then have *snd* (*execute tmL1-2* (*0*, *?tps*) (*Suc i*)) =
tpsL t xs (*Suc i*) *q mvs* (λ*j*. *if exec t* <#> *j* < *Suc i then exec t* <.> *j else* □)
by *simp*
moreover have *esL1-2* ! *i* = (*n* + *1*, *Suc i*)
unfolding *esL1-2-def*
using *True nth-append*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-eq-plus1 add.commute add-Suc diff-zero length-map length-upt nth-map-upt*)
ultimately show *?thesis*
using *tpsL-pos-0 tpsL-pos-1* **by** *simp*
next
case *False*
then have *i* = *TT*
using *len that* **by** *simp*
then have *execute tmL1-2* (*0*, *?tps*) (*Suc i*) = (*1*, *tpsL t xs TT q mvs* (λ*j*. *exec t* <.> *j*))
using *execute-cmdL2-Suc-TT assms* **by** *simp*
moreover have *esL1-2* ! *i* = (*n* + *1*, *TT*)
using ‹*i* = *TT*› *esL1-2-def* **by** (*metis* (*no-types*, *lifting*) *diff-zero length-map length-upt nth-append-length*)
ultimately show *?thesis*
using *tpsL-pos-0 tpsL-pos-1* **by** *auto*
qed
qed

definition *esL2 t* ≡ *esL1 t* @ *esL1-2*

definition *tpsL2 t* ≡ *tpsL t* (*replicate k* (*0*, *Some 0*)) *TT 0* (λ-. □) (λ*j*. *exec t* <.> *j*)

lemma *tmL2*:
assumes *t* < *TT*
shows *traces tmL2* (*tpsL0 t*) (*esL2 t*) (*tpsL2 t*)
unfolding *tmL2-def esL2-def*
proof (*rule traces-sequential*[*OF tmL1*])
show *traces tmL1-2* (*tpsL1 t*) *esL1-2* (*tpsL2 t*)
using *traces-tmL1-2*[*OF - assms*] **by** (*simp add*: *tpsL1-def tpsL2-def*)
qed

definition *sim-nextstate t* ≡
(*if fst* (*exec t*) < *length M*
then fst ((*M* ! (*fst* (*exec t*))) (*config-read* (*exec t*)))
else fst (*exec t*))

lemma *sim-nextstate*: *fst* (*exec* (*Suc t*)) = *sim-nextstate t*
proof (*cases fst* (*exec t*) < *length M*)
case *True*
let *?cfg* = *exec t*
let *?rs* = *config-read ?cfg*
let *?cmd* = *M* ! (*fst ?cfg*)
have *exec* (*Suc t*) = *sem ?cmd ?cfg*
using *exec-def True* **by** (*simp add*: *exe-lt-length*)
then have *2*: *fst* (*exec* (*Suc t*)) = *fst* (*sem ?cmd ?cfg*)
by *simp*
also have ... = *fst* (*?cmd ?rs*)
using *sem′* **by** *simp*

also have ... = *sim-nextstate t*
using *sim-nextstate-def True* **by** *simp*
finally show *?thesis* **.**
next
case *False*
then show *?thesis*
using *exec-def exe-def sim-nextstate-def* **by** *simp*
qed

definition *sim-write t* ≡
(*if fst* (*exec t*) < *length M*
then map fst (*snd* ((*M* ! (*fst* (*exec t*))) (*config-read* (*exec t*))))
else config-read (*exec t*))

lemma *sim-write-nth*:
assumes *fst* (*exec t*) < *length M* **and** *j* < *k*
shows *sim-write t* ! *j* = ((*M* ! (*fst* (*exec t*))) (*config-read* (*exec t*)) [.] *j*)
proof −
have *length* (*snd* ((*M* ! (*fst* (*exec t*))) (*config-read* (*exec t*)))) = *k*
using *assms turing-commandD(1) exec-def execute-num-tapes read-length start-config-length tm-M turing-machine-def*
by (*metis add-gr-0 less-imp-add-positive nth-mem*)
then show *?thesis*
using *sim-write-def assms* **by** *simp*
qed

lemma *sim-write-nth-else*:
assumes ¬ (*fst* (*exec t*) < *length M*)
shows *sim-write t* ! *j* = *config-read* (*exec t*) ! *j*
by (*simp add*: *assms sim-write-def*)

lemma *sim-write-nth-less-G*:
assumes *j* < *k*
shows *sim-write t* ! *j* < *G*
proof (*cases fst* (*exec t*) < *length M*)
case *True*
then have ∗: *sim-write t* ! *j* = (*M* ! (*fst* (*exec t*))) (*config-read* (*exec t*)) [.] *j*
using *sim-write-nth assms* **by** *simp*
have *turing-command k* (*length M*) *G* (*M* ! (*fst* (*exec t*)))
using *tm-M True turing-machineD(3)* **by** *simp*
moreover have ∀ *i*<*k*. (*config-read* (*exec t*)) ! *i* < *G*
using *read-alphabet exec-def tm-M* **by** (*simp add*: *zs-less-G*)
moreover have *length* (*config-read* (*exec t*)) = *k*
by (*metis Suc-1 exec-def execute-num-tapes start-config-length less-le-trans read-length*
turing-machine-def tm-M zero-less-Suc)
ultimately have (*M* ! (*fst* (*exec t*))) (*config-read* (*exec t*)) [.] *j* < *G*
using *assms exec-def turing-commandD(2)* **by** *simp*
then show *?thesis*
using ∗ **by** *simp*
next
case *False*
then show *?thesis*
using *exec-def sim-write-nth-else assms tape-alphabet*
by (*simp add*: *read-alphabet tm-M zs-less-G*)
qed

lemma *sim-write*:
assumes *j* < *k*
shows *exec* (*Suc t*) <:> *j* = (*exec t* <:> *j*)(*exec t* <#> *j* := *sim-write t* ! *j*)
proof (*cases fst* (*exec t*) < *length M*)
case *True*
let *?cfg* = *exec t*
let *?rs* = *config-read ?cfg*
let *?cmd* = *M* ! (*fst ?cfg*)

```
  have len-rs: length ?rs = k
    using assms exec-def execute-num-tapes start-config-length read-length tm-M by simp
  have turing-command k (length M) G ?cmd
    using True tm-M turing-machineD(3) by simp
  then have len: length (snd (?cmd ?rs)) = k
    using len-rs turing-commandD(1) by simp

  have sim-write t = map fst (snd (?cmd ?rs))
    using sim-write-def True by simp
  then have 1: sim-write t ! j = ?cmd ?rs [.] j
    by (simp add: assms len)

  have 2: exec (Suc t) = sem ?cmd ?cfg
    using exec-def True by (simp add: exe-lt-length)

  have snd (sem ?cmd ?cfg) = map (λ(a, tp). act a tp) (zip (snd (?cmd ?rs)) (snd ?cfg))
    using sem' by simp
  then have snd (sem ?cmd ?cfg) ! j = (λ(a, tp). act a tp) ((snd (?cmd ?rs) ! j), (snd ?cfg ! j))
    using len assms len-rs read-length by simp
  then have sem ?cmd ?cfg <!> j = act (snd (?cmd ?rs) ! j) (?cfg <!> j)
    by simp
  then have sem ?cmd ?cfg <:> j = fst (act (snd (?cmd ?rs) ! j) (?cfg <!> j))
    by simp
  then have sem ?cmd ?cfg <:> j = (?cfg <:> j)(?cfg <#> j := fst (snd (?cmd ?rs) ! j))
    using act by simp
  then have sem ?cmd ?cfg <:> j = (?cfg <:> j)(?cfg <#> j := ?cmd ?rs [.] j) .
  then show ?thesis
    using 1 2 by simp
next
  case False
  let ?cfg = exec t
  let ?rs = config-read ?cfg
  have len-rs: length ?rs = k
    using assms exec-def execute-num-tapes start-config-length read-length tm-M by simp
  then have 1: sim-write t ! j = ?rs ! j
    using False by (simp add: sim-write-def)

  have 2: ?rs ! j = exec t <.> j
    using assms len-rs read-abbrev read-length by auto

  have exec (Suc t) = exec t
    using exec-def exe-def False by simp
  then have exec (Suc t) <:> j = exec t <:> j
    by simp

  then show ?thesis
    using 1 2 by simp
qed

corollary sim-write':
  assumes j < k
  shows (exec (Suc t) <:> j) (exec t <#> j) = sim-write t ! j
  using assms sim-write by simp

definition sim-move t ≡ map direction-to-symbol
  (if fst (exec t) < length M
   then map snd (snd ((M ! (fst (exec t))) (config-read (exec t))))
   else replicate k Stay)

lemma sim-move-nth:
  assumes fst (exec t) < length M and j < k
  shows sim-move t ! j = direction-to-symbol ((M ! (fst (exec t))) (config-read (exec t)) [~] j)
proof -
```

```
  have k = ||execute M (start-config k zs) t||
   by (metis (no-types) Suc-1 execute-num-tapes start-config-length less-le-trans tm-M turing-machine-def zero-less-Suc)
  then show ?thesis
    by (smt (verit, best) turing-commandD(1) assms(1,2) exec-def length-map nth-map read-length sim-move-def
tm-M turing-machineD(3))
qed

lemma sim-move-nth-else:
  assumes ¬ (fst (exec t) < length M) and j < k
  shows sim-move t ! j = 1
  using assms sim-move-def direction-to-symbol-def by simp

lemma sim-move:
  assumes j < k
  shows exec (Suc t) <#> j = exec t <#> j + sim-move t ! j − 1
proof (cases fst (exec t) < length M)
  case True
  let ?cfg = exec t
  let ?rs = config-read ?cfg
  let ?cmd = M ! (fst ?cfg)
  have len-rs: length ?rs = k
    using assms exec-def execute-num-tapes start-config-length read-length tm-M by simp
  have turing-command k (length M) G ?cmd
    using True tm-M turing-machineD(3) by simp
  then have len: length (snd (?cmd ?rs)) = k
    using len-rs turing-commandD(1) by simp

  have 1: sim-move t ! j = direction-to-symbol (?cmd ?rs [~] j)
    using assms sim-move-nth True by simp

  have exec (Suc t) = sem ?cmd ?cfg
    using exec-def True by (simp add: exe-lt-length)
  then have 2: exec (Suc t) <#> j = sem ?cmd ?cfg <#> j
    by simp

  have snd (sem ?cmd ?cfg) = map (λ(a, tp). act a tp) (zip (snd (?cmd ?rs)) (snd ?cfg))
    using sem′ by simp
  then have snd (sem ?cmd ?cfg) ! j = (λ(a, tp). act a tp) ((snd (?cmd ?rs) ! j), (snd ?cfg ! j))
    using len assms len-rs read-length by simp
  then have sem ?cmd ?cfg <!> j = act (snd (?cmd ?rs) ! j) (?cfg <!> j)
    by simp
  then have sem ?cmd ?cfg <#> j = snd (act (snd (?cmd ?rs) ! j) (?cfg <!> j))
    by simp
  then have sem ?cmd ?cfg <#> j =
     (case ?cmd ?rs [~] j of Left ⇒ ?cfg <#> j − 1 | Stay ⇒ ?cfg <#> j | Right ⇒ ?cfg <#> j + 1)
    using act by simp
  then show ?thesis
    using 1 2 direction-to-symbol-def by (cases ?cmd ?rs [~] j) simp-all
next
  case False
  then have exec (Suc t) <#> j = exec t <#> j
    using exec-def exe-def by simp
  moreover have sim-move t ! j = 1
    using direction-to-symbol-def sim-move-def assms False by simp
  ultimately show ?thesis
    by simp
qed

definition tpsL3 t ≡ tpsL
  t
  (replicate k (0, Some 0))
  TT
  1
```

```
  (λj. sim-move t ! j)
  (λj. sim-write t ! j)

lemma read-execute: config-read (exec t) = map (λj. (exec t) <.> j) [0..<k]
  (is ?lhs = ?rhs)
proof (rule nth-equalityI)
  have len: length ?lhs = k
    by (metis Suc-1 exec-def execute-num-tapes start-config-length less-le-trans read-length
      tm-M turing-machine-def zero-less-Suc)
  then show length ?lhs = length ?rhs
    by simp
  show ?lhs ! i = ?rhs ! i if i < length ?lhs for i
    using len read-abbrev read-length that by auto
qed

lemma sem-cmdL3: sem cmdL3 (0, tpsL2 t) = (1, tpsL3 t)
proof (rule semI[of 2 * k + 3])
  show proper-command (2 * k + 3) cmdL3
    using cmdL3-def by simp
  show len: length (tpsL2 t) = 2 * k + 3
    using tpsL2-def by simp
  show length (tpsL3 t) = 2 * k + 3
    using tpsL3-def by simp
  show fst (cmdL3 (read (tpsL2 t))) = 1
    by (simp add: cmdL3-def)
  show act (cmdL3 (read (tpsL2 t)) [!] j) (tpsL2 t ! j) = tpsL3 t ! j if j < 2 * k + 3 for j
  proof -
    define rs where rs = read (tpsL2 t)
    then have rs2: rs ! 2 = fst (exec t)
      using tpsL2-def read-tpsL-2 by simp
    have drop (k + 3) rs = map (λj. exec t <.> j) [0..<k] (is ?lhs = ?rhs)
    proof (rule nth-equalityI)
      show length ?lhs = length ?rhs
        using rs-def read-length tpsL2-def by simp
      then have len-lhs: length ?lhs = k
        by simp
      show ?lhs ! i = ?rhs ! i if i < length ?lhs for i
      proof -
        have i < k
          using that len-lhs by simp
        have length rs = 2 * k + 3
          using rs-def read-length tpsL2-def by simp
        then have ?lhs ! i = rs ! (i + k + 3)
          by (simp add: add.assoc add.commute)
        also have ... = exec t <.> i
          unfolding rs-def tpsL2-def using read-tpsL-4[of i + k + 3] ‹i < k› by simp
        finally show ?thesis
          using ‹i < k› by simp
      qed
    qed
    then have drop: drop (k + 3) rs = config-read (exec t)
      using read-execute by simp
    then have drop-less-G: ∀h∈set (drop (k + 3) rs). h < G
      using exec-def tm-M read-alphabet-set zs-less-G by presburger

    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
      using ‹j < 2 * k + 3› by linarith
    then show ?thesis
    proof (cases)
      case 1
      then have cmdL3 rs [!] j = (rs ! 0, Stay)
        using cmdL3-def by simp
      moreover have tpsL2 t ! j = (⌊zs⌋, n + 1)
```

    using *tpsL2-def 1* **by** (*simp add*: *tpsL-0*)
  moreover have *tpsL3 t* ! *j* = (⌊*zs*⌋, *n* + *1*)
    using *tpsL3-def 1* **by** (*simp add*: *tpsL-0*)
  ultimately show *?thesis*
    using *act-Stay* **by** (*metis 1 len rs-def that*)
next
  case *2*
  then have *cmdL3 rs* [!] *j* = (*rs* ! *1*, *Stay*)
    using *cmdL3-def* **by** *simp*
  moreover have *tpsL2 t* ! *j* = (*zip-cont t* (*replicate k* (*0*, *Some 0*)), *TT*)
    using *tpsL2-def 2 tpsL-1* **by** *simp*
  moreover have *tpsL3 t* ! *j* = (*zip-cont t* (*replicate k* (*0*, *Some 0*)), *TT*)
    using *tpsL3-def 2 tpsL-1* **by** *simp*
  ultimately show *?thesis*
    using *act-Stay* **by** (*metis 2 len rs-def that*)
next
  case *3*
  show *?thesis*
  proof (*cases rs* ! *2* < *length M*)
    case *True*
    then have *cmdL3 rs* [!] *j* = (*fst* ((*M* ! (*rs* ! *2*)) (*drop* (*k* + *3*) *rs*)), *Stay*)
      using *3 drop-less-G cmdL3-at-2a* **by** *simp*
    also have ... = (*fst* ((*M* ! (*rs* ! *2*)) (*config-read* (*exec t*))), *Stay*)
      using *drop* **by** *simp*
    also have ... = (*fst* (*exec* (*Suc t*)), *Stay*)
      using *True rs2 sim-nextstate sim-nextstate-def* **by** *auto*
    finally have *cmdL3 rs* [!] *j* = (*fst* (*exec* (*Suc t*)), *Stay*) **.**
    then show *?thesis*
      using *tpsL2-def tpsL3-def tpsL-def 3 act-onesie rs-def* **by** *simp*
  next
    case *False*
    then have *cmdL3 rs* [!] *j* = (*rs* ! *2*, *Stay*)
      using *3 cmdL3-at-2b* **by** *simp*
    moreover have *fst* (*exec* (*Suc t*)) = *rs* ! *2*
      using *False rs2 sim-nextstate sim-nextstate-def* **by** *auto*
    ultimately have *cmdL3 rs* [!] *j* = (*fst* (*exec* (*Suc t*)), *Stay*)
      by *simp*
    then show *?thesis*
      using *tpsL2-def tpsL3-def tpsL-def 3 act-onesie rs-def* **by** *simp*
  qed
next
  case *4*
  then have *1*: *j* − *3* < *k*
    by *auto*
  have *2*: *tpsL2 t* ! *j* = ⌈□⌉
    using *tpsL2-def tpsL-mvs′ 4* **by** *simp*
  have *3*: *tpsL3 t* ! *j* = ⌈*sim-move t* ! (*j* − *3*)⌉
    using *tpsL3-def tpsL-mvs′ 4* **by** *simp*
  show *?thesis*
  proof (*cases rs* ! *2* < *length M*)
    case *True*
    then have *cmdL3 rs* [!] *j* = (*direction-to-symbol* ((*M* ! (*rs* ! *2*)) (*drop* (*k* + *3*) *rs*) [~] (*j* − *3*)), *Stay*)
      using *cmdL3-at-3a 4 drop-less-G* **by** *simp*
    also have ... = (*direction-to-symbol* ((*M* ! (*fst* (*exec t*))) (*config-read* (*exec t*)) [~] (*j* − *3*)), *Stay*)
      using *drop True rs2* **by** *simp*
    also have ... = (*sim-move t* ! (*j* − *3*), *Stay*)
      using *sim-move-nth True 1 rs2* **by** *simp*
    finally have *cmdL3 rs* [!] *j* = (*sim-move t* ! (*j* − *3*), *Stay*) **.**
    then show *?thesis*
      using *2 3 act-onesie* **by** (*simp add*: *rs-def*)
  next
    case *False*
    then have *cmdL3 rs* [!] *j* = (*1*, *Stay*)

```
          using cmdL3-at-3b 4 by simp
        then have cmdL3 rs [!] j = (sim-move t ! (j − 3), Stay)
          using sim-move-nth-else False 1 rs2 by simp
        then show ?thesis
          using 2 3 act-onesie by (simp add: rs-def)
      qed
    next
      case 5
      then have 1: j − k − 3 < k
        by auto
      have 2: tpsL2 t ! j = ⌈exec t <.> (j − k − 3)⌉
        using tpsL2-def tpsL-symbs' 5 by simp
      have 3: tpsL3 t ! j = ⌈sim-write t ! (j − k − 3)⌉
        using tpsL3-def tpsL-symbs' 5 by simp
      show ?thesis
      proof (cases rs ! 2 < length M)
        case True
        then have cmdL3 rs [!] j = ((M ! (rs ! 2)) (drop (k + 3) rs) [.] (j − k − 3), Stay)
          using 5 cmdL3-at-4a drop-less-G by simp
        also have ... = ((M ! (fst (exec t))) (config-read (exec t)) [.] (j − k − 3), Stay)
          using drop True rs2 by simp
        also have ... = (sim-write t ! (j − k − 3), Stay)
          using sim-write-nth 5 rs2 True by auto
        finally have cmdL3 rs [!] j = (sim-write t ! (j − k − 3), Stay) .
        then show ?thesis
          using 2 3 act-onesie rs-def by simp
      next
        case False
        then have cmdL3 rs [!] j = (rs ! j, Stay)
          using 5 cmdL3-at-4b by simp
        also have ... = (exec t <.> (j − k − 3), Stay)
          using tpsL2-def read-tpsL-4 5 rs-def by simp
        also have ... = (config-read (exec t) ! (j − k − 3), Stay)
        proof −
          have length [k..<j] − 3 < k
            by (metis 1 length-upt)
          then show ?thesis
            using ‹drop (k + 3) rs = map (λj. exec t <.> j) [0..<k]› drop by auto
        qed
        also have ... = (sim-write t ! (j − k − 3), Stay)
          using sim-write-nth-else False rs2 by simp
        finally have cmdL3 rs [!] j = (sim-write t ! (j − k − 3), Stay) .
        then show ?thesis
          using 2 3 act-onesie rs-def by simp
      qed
    qed
  qed
qed

lemma execute-tmL2-3: execute tmL2-3 (0, tpsL2 t) 1 = (1, tpsL3 t)
proof −
  have execute tmL2-3 (0, tpsL2 t) 1 = exe tmL2-3 (execute tmL2-3 (0, tpsL2 t) 0)
    by simp
  also have ... = exe tmL2-3 (0, tpsL2 t)
    by simp
  also have ... = sem cmdL3 (0, tpsL2 t)
    using tmL2-3-def by (simp add: exe-lt-length)
  finally show ?thesis
    using sem-cmdL3 by simp
qed

definition esL3 t ≡ esL2 t @ [(n + 1, TT)]
```

```
lemma tmL3:
  assumes t < TT
  shows traces tmL3 (tpsL0 t) (esL3 t) (tpsL3 t)
  unfolding tmL3-def esL3-def
proof (rule traces-sequential[OF tmL2[OF assms]])
  show traces tmL2-3 (tpsL2 t) [(n + 1, Suc (fmt n))] (tpsL3 t)
  proof
    let ?es = [(n + 1, TT)]
    show execute tmL2-3 (0, tpsL2 t) (length ?es) = (length tmL2-3, tpsL3 t)
      using tmL2-3-def execute-tmL2-3 by simp
    show ⋀i. i < length ?es ⟹ fst (execute tmL2-3 (0, tpsL2 t) i) < length tmL2-3
      using tmL2-3-def execute-tmL2-3 by simp
    show (execute tmL2-3 (0, tpsL2 t) (Suc i)) <#> 0 = fst (?es ! i) ∧
        (execute tmL2-3 (0, tpsL2 t) (Suc i)) <#> 1 = snd (?es ! i)
      if i < length ?es for i
      using execute-tmL2-3 that tpsL3-def tpsL-pos-0 tpsL-pos-1
      by (metis One-nat-def fst-conv length-Cons less-one list.size(3) nth-Cons-0 snd-conv)
  qed
qed

definition esL4 t ≡ esL3 t @ map (λi. (n + 1, i)) (rev [0..<TT]) @ [(n + 1, 0)]

lemma len-esL4: length (esL4 t) = t + 2 ∗ TT + 4
  using esL4-def esL3-def esL2-def esL1-def esL1-2-def by simp

definition tpsL4 t ≡ tpsL
  t
  (replicate k (0, Some 0))
  0
  1
  (λj. sim-move t ! j)
  (λj. sim-write t ! j)

lemma tmL4:
  assumes t < TT
  shows traces tmL4 (tpsL0 t) (esL4 t) (tpsL4 t)
  unfolding tmL4-def esL4-def
proof (rule traces-sequential[where ?tps2.0=tpsL3 t])
  show traces tmL3 (tpsL0 t) (esL3 t) (tpsL3 t) using tmL3 assms .
  show traces tm-left-until1 (tpsL3 t) (map (Pair (n + 1)) (rev [0..<TT]) @ [(n + 1, 0)]) (tpsL4 t)
  proof (rule traces-tm-left-until-1I)
    show 1 < length (tpsL3 t)
      using tpsL3-def by simp
    show begin-tape {y. y < G ^ (2 ∗ k + 2) + 2 ∧ 1 < y ∧ dec y ! (2 ∗ k + 1) = 1} (tpsL3 t ! 1)
      using begin-tape-zip-cont tpsL3-def tpsL-def by simp
    show map (Pair (n + 1)) (rev [0..<TT]) @ [(n + 1, 0)] =
        map (Pair (tpsL3 t :#: 0)) (rev [0..<tpsL3 t :#: 1]) @ [(tpsL3 t :#: 0, 0)]
      using tpsL3-def tpsL-def by simp
    show tpsL4 t = (tpsL3 t)[1 := tpsL3 t ! 1 |#=| 0]
      using tpsL3-def tpsL4-def tpsL-def by simp
  qed
qed

lemma enc-upd-zip-cont-None-Some:
  assumes jj < k
    and length xs = k
    and xs ! jj = (1, None)
    and i = (exec (Suc t) <#> jj)
  shows enc-upd (zip-cont t xs i) (k + jj) 1 = zip-cont t (xs[jj:=(1, Some 1)]) i
proof −
  have i < TT
    using assms(1,4) exec-pos-less-TT by metis
```

let *?n = zip-cont t xs i*
let *?xs = xs[jj:=(1, Some 1)]*
have *zip-cont t ?xs i = enc*
(map (λj. (exec (t + fst (?xs ! j)) <:> j) i) [0..<k] @
map (λj. case snd (?xs ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k] @
[if i < t then 1 else 0,
if i = 0 then 1 else 0])
(**is** - = *enc ?ys*)
using *zip-cont-def ‹i < TT›* **by** *simp*
moreover have *?ys = (dec ?n) [k+jj:=1]*
proof (*rule nth-equalityI*)
show *len: length ?ys = length ((dec ?n) [k+jj:=1])*
by (*simp add: ‹i < TT› dec-zip-cont*)
have *lenys: length ?ys = 2 ∗ k + 2*
by *simp*
show *?ys ! j = (dec ?n) [k+jj:=1] ! j* **if** *j < length ?ys* **for** *j*
proof −
consider *j < k | k ≤ j ∧ j < 2 ∗ k | j = 2 ∗ k | j = 2 ∗ k + 1*
using *lenys ‹j < length ?ys›* **by** *linarith*
then show *?thesis*
proof (*cases*)
case *1*
then have *?ys ! j = (exec (t + fst (?xs ! j)) <:> j) i*
using *assms(2)* **by** (*simp add: less-k-nth*)
have (*dec ?n*) *[k+jj:=1] ! j = dec ?n ! j*
using *1* **by** *simp*
also have ... = (*exec (t + fst (xs ! j)) <:> j) i*
by (*simp add: 1 ‹i < TT› dec-zip-cont-less-k*)
also have ... = (*exec (t + fst (?xs ! j)) <:> j) i*
using *assms(1−3)* **by** (*metis fst-eqD nth-list-update*)
also have ... = *?ys ! j*
using *assms(2) 1* **by** (*simp add: less-k-nth*)
finally show *?thesis*
by *simp*
next
case *2*
show *?thesis*
proof (*cases j = k + jj*)
case *True*
then have *?ys ! j = (case snd (?xs ! jj) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> jj then 1 else 0)*
using *assms(2) 2* **by** (*simp add: less-2k-nth*)
also have ... = (*if i = exec (t + 1) <#> jj then 1 else 0*)
by (*simp add: assms(1) assms(2)*)
also have ... = *1*
using *assms(1,4)* **by** *simp*
finally have *?ys ! j = 1* **.**
moreover have (*dec ?n*) *[k+jj:=1] ! j = 1*
using *True len that* **by** *simp*
ultimately show *?thesis*
by *simp*
next
case *False*
then have ∗: *?xs ! (j − k) = xs ! (j − k)*
by (*metis 2 add-diff-inverse-nat le-imp-less-Suc not-less-eq nth-list-update-neq*)
have *?ys ! j =*
(*case snd (?xs ! (j − k)) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> (j − k) then 1 else 0*)
using *assms(2) 2* **by** (*simp add: less-2k-nth*)
have (*dec ?n*) *[k+jj:=1] ! j = (dec ?n) ! j*
using *2 False* **by** *simp*
also have ... =
(*case snd (xs ! (j − k)) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> (j − k) then 1 else 0*)
using *2 ‹i < TT› dec-zip-cont-less-2k* **by** *simp*

```
          also have ... = (case snd (?xs ! (j − k)) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> (j − k)
then 1 else 0)
            using ∗ by simp
          also have ... = ?ys ! j
            using assms(2) 2 by (simp add: less-2k-nth)
          finally show ?thesis
            by simp
        qed
      next
        case 3
        then have ?ys ! j = (if i < t then 1 else 0)
          by (simp add: twok-nth)
        moreover have (dec ?n) [k+jj:=1] ! j = (if i < t then 1 else 0)
            using 3 assms(1) dec-zip-cont-2k ‹i < TT› by simp
        ultimately show ?thesis
          by simp
      next
        case 4
        then have ?ys ! j = (if i = 0 then 1 else 0)
          using twok1-nth by fast
        moreover have (dec ?n) [k+jj:=1] ! j = (if i = 0 then 1 else 0)
            using 4 assms(1) dec-zip-cont-2k1 ‹i < TT› by simp
        ultimately show ?thesis
          by simp
      qed
    qed
  qed
  ultimately show ?thesis
    using enc-upd-def by simp
qed

lemma enc-upd-zip-cont-None-Some-Right:
  assumes jj < k
    and length xs = k
    and xs ! jj = (1, None)
    and i = Suc (exec t <#> jj)
    and sim-move t ! jj = 2
  shows enc-upd (zip-cont t xs i) (k + jj) 1 = zip-cont t (xs[jj:=(1, Some 1)]) i
proof −
  have i = (exec (Suc t) <#> jj)
    using assms sim-move by simp
  then show ?thesis
    using enc-upd-zip-cont-None-Some[OF assms(1−3)] by simp
qed

lemma enc-upd-zip-cont-None-Some-Left:
  assumes jj < k
    and length xs = k
    and xs ! jj = (1, None)
    and Suc i = exec t <#> jj
    and sim-move t ! jj = 0
  shows enc-upd (zip-cont t xs i) (k + jj) 1 = zip-cont t (xs[jj:=(1, Some 1)]) i
proof −
  have i = (exec (Suc t) <#> jj)
    using assms sim-move by simp
  then show ?thesis
    using enc-upd-zip-cont-None-Some[OF assms(1−3)] by simp
qed

lemma enc-upd-zip-cont-Some-None:
  assumes jj < k
    and length xs = k
    and xs ! jj = (1, Some 0)
```

```
    and i = exec t <#> jj
  shows enc-upd (zip-cont t xs i) (k + jj) 0 = zip-cont t (xs[jj:=(1, None)]) i
proof −
  have i < TT
    using assms(1,4) by (simp add: exec-pos-less-TT)
  let ?n = zip-cont t xs i
  let ?xs = xs[jj:=(1, None)]
  have zip-cont t ?xs i = enc
     (map (λj. (exec (t + fst (?xs ! j)) <:> j) i) [0..<k] @
      map (λj. case snd (?xs ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k] @
      [if i < t then 1 else 0,
       if i = 0 then 1 else 0])
       (is - = enc ?ys)
    using zip-cont-def ‹i < TT› by simp
  moreover have ?ys = (dec ?n) [k+jj:=0]
  proof (rule nth-equalityI)
    show len: length ?ys = length ((dec ?n) [k+jj:=0])
      by (simp add: ‹i < TT› dec-zip-cont)
    have lenys: length ?ys = 2 ∗ k + 2
      by simp
    show ?ys ! j = (dec ?n) [k+jj:=0] ! j if j < length ?ys for j
    proof −
      consider j < k | k ≤ j ∧ j < 2 ∗ k | j = 2 ∗ k | j = 2 ∗ k + 1
        using lenys ‹j < length ?ys› by linarith
      then show ?thesis
      proof (cases)
        case 1
        then have ?ys ! j = (exec (t + fst (?xs ! j)) <:> j) i
          using assms(2) by (simp add: less-k-nth)
        have (dec ?n) [k+jj:=0] ! j = dec ?n ! j
          using 1 by simp
        also have ... = (exec (t + fst (xs ! j)) <:> j) i
          by (simp add: 1 ‹i < TT› dec-zip-cont-less-k)
        also have ... = (exec (t + fst (?xs ! j)) <:> j) i
          using assms(1−3) by (metis fst-eqD nth-list-update)
        also have ... = ?ys ! j
          using assms(2) 1 by (simp add: less-k-nth)
        finally show ?thesis
          by simp
      next
        case 2
        show ?thesis
        proof (cases j = k + jj)
          case True
          then have ?ys ! j = (case snd (?xs ! jj) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> jj then 1
else 0)
            using assms(2) 2 by (simp add: less-2k-nth)
          then have ?ys ! j = 0
            using assms(1,2) by simp
          moreover have (dec ?n) [k+jj:=0] ! j = 0
            using True len that by simp
          ultimately show ?thesis
            by simp
        next
          case False
          then have ∗: ?xs ! (j − k) = xs ! (j − k)
            by (metis 2 add-diff-inverse-nat le-imp-less-Suc not-less-eq nth-list-update-neq)
          have ?ys ! j =
              (case snd (?xs ! (j − k)) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> (j − k) then 1 else 0)
            using assms(2) 2 by (simp add: less-2k-nth)
          have (dec ?n) [k+jj:=0] ! j = (dec ?n) ! j
            using 2 False by simp
          also have ... =
```

(*case snd* (*xs* ! (*j* − *k*)) *of None* ⇒ *0* | *Some d* ⇒ *if i* = *exec* (*t* + *d*) <#> (*j* − *k*) *then 1 else 0*)
using *2* ‹*i* < *TT*› *dec-zip-cont-less-2k* **by** *simp*
also have ... = (*case snd* (*?xs* ! (*j* − *k*)) *of None* ⇒ *0* | *Some d* ⇒ *if i* = *exec* (*t* + *d*) <#> (*j* − *k*) *then 1 else 0*)
using ∗ **by** *simp*
also have ... = *?ys* ! *j*
using *assms*(*2*) *2* **by** (*simp add*: *less-2k-nth*)
finally show *?thesis*
by *simp*
qed
next
case *3*
then have *?ys* ! *j* = (*if i* < *t then 1 else 0*)
by (*simp add*: *twok-nth*)
moreover have (*dec ?n*) [*k*+*jj*:=*0*] ! *j* = (*if i* < *t then 1 else 0*)
using *3 assms*(*1*) *dec-zip-cont-2k* ‹*i* < *TT*› **by** *simp*
ultimately show *?thesis*
by *simp*
next
case *4*
then have *?ys* ! *j* = (*if i* = *0 then 1 else 0*)
using *twok1-nth* **by** *fast*
moreover have (*dec ?n*) [*k*+*jj*:=*0*] ! *j* = (*if i* = *0 then 1 else 0*)
using *4 assms*(*1*) *dec-zip-cont-2k1* ‹*i* < *TT*› **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed
qed
ultimately show *?thesis*
using *enc-upd-def* **by** *simp*
qed

lemma *zip-cont-nth-eq-updI1*:
assumes *i* < *TT*
and *jj* < *k*
and *length xs* = *k*
and *xs* ! *jj* = (*0*, *Some 0*)
and (*exec* (*Suc t*) <:> *jj*) *i* = *u*
shows *enc-upd* (*zip-cont t xs i*) *jj u* = *zip-cont t* (*xs*[*jj*:=(*1*, *Some 0*)]) *i*
proof −
let *?n* = *zip-cont t xs i*
let *?xs* = *xs*[*jj*:=(*1*, *Some 0*)]
have *zip-cont t ?xs i* = *enc*
(*map* (λ*j*. (*exec* (*t* + *fst* (*?xs* ! *j*)) <:> *j*) *i*) [*0*..<*k*] @
map (λ*j*. *case snd* (*?xs* ! *j*) *of None* ⇒ *0* | *Some d* ⇒ *if i* = *exec* (*t* + *d*) <#> *j then 1 else 0*) [*0*..<*k*] @
[*if i* < *t then 1 else 0*,
if i = *0 then 1 else 0*])
(**is** - = *enc ?ys*)
using *zip-cont-def assms*(*1*) **by** *simp*
moreover have *?ys* = (*dec ?n*) [*jj*:=*u*]
proof (*rule nth-equalityI*)
show *len-eq*: *length ?ys* = *length* ((*dec ?n*) [*jj*:=*u*])
using *assms*(*1*) *dec-zip-cont two-tape-axioms zs-proper zs-less-G* **by** *simp*
have *lenys*: *length ?ys* = *2* ∗ *k* + *2*
by *simp*
show *?ys* ! *j* = (*dec ?n*) [*jj*:=*u*] ! *j* **if** *j* < *length ?ys* **for** *j*
proof −
consider *j* < *k* | *k* ≤ *j* ∧ *j* < *2* ∗ *k* | *j* = *2* ∗ *k* | *j* = *2* ∗ *k* + *1*
using *lenys* ‹*j* < *length ?ys*› **by** *linarith*
then show *?thesis*
proof (*cases*)
case *1*

```
  then have *: ?ys ! j = (exec (t + fst (?xs ! j)) <:> j) i
    by (simp add: less-k-nth)
  show ?thesis
  proof (cases j = jj)
    case True
    then have ?ys ! j = (exec (Suc t) <:> j) i
      using 1 assms(3) * by simp
    moreover have (dec ?n) [jj:=u] ! j = u
      using True len-eq that by simp
    ultimately show ?thesis
      using assms(5) True by simp
  next
    case False
    then have (dec ?n) [jj:=u] ! j = (exec (t + fst (xs ! j)) <:> j) i
      using dec-zip-cont-less-k 1 assms(1) by simp
    moreover have ?ys ! j = (exec (t + fst (xs ! j)) <:> j) i
      using False * by simp
    ultimately show ?thesis
      by simp
  qed
next
  case 2
  then have *: ?ys ! j =
      (case snd (?xs ! (j − k)) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> (j − k) then 1 else 0)
    by (simp add: less-2k-nth)
  show ?thesis
  proof (cases j = k + jj)
    case True
    then have j − k = jj
      by simp
    then have snd (?xs ! (j − k)) = Some 0
      using assms(2,3) by simp
    then have lhs: ?ys ! j = (if i = exec t <#> jj then 1 else 0)
      using * True by simp
    have (dec ?n) [jj:=u] ! j = (dec ?n) ! j
      using True assms(2) by simp
    also have ... =
        (case snd (xs ! (j − k)) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> (j − k) then 1 else 0)
      using True 2 assms(1,4) dec-zip-cont-less-2k by simp
    also have ... = (if i = exec t <#> jj then 1 else 0)
      using True assms(4) by simp
    finally have (dec ?n) [jj:=u] ! j = (if i = exec t <#> jj then 1 else 0) .
    then show ?thesis
      using lhs True by simp
  next
    case False
    then have j − k ≠ jj
      using 2 by auto
    then have snd (?xs ! (j − k)) = snd (xs ! (j − k))
      by simp
    then have lhs: ?ys ! j =
        (case snd (xs ! (j − k)) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> (j − k) then 1 else 0)
      using * by simp
    have (dec ?n) [jj:=u] ! j = (dec ?n) ! j
      using 2 assms(2) by simp
    then have (dec ?n) [jj:=u] ! j =
        (case snd (xs ! (j − k)) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> (j − k) then 1 else 0)
      using False 2 assms(1) dec-zip-cont-less-2k by simp
    then show ?thesis
      using lhs by simp
  qed
next
  case 3
```

```
        then have ?ys ! j = (if i < t then 1 else 0)
          by (simp add: twok-nth)
        moreover have (dec ?n) [jj:=u] ! j = (if i < t then 1 else 0)
            using 3 assms(1,2) dec-zip-cont-2k by simp
        ultimately show ?thesis
          by simp
      next
        case 4
        then have ?ys ! j = (if i = 0 then 1 else 0)
          using twok1-nth by fast
        moreover have (dec ?n) [jj:=u] ! j = (if i = 0 then 1 else 0)
            using 4 assms(1,2) dec-zip-cont-2k1 by simp
        ultimately show ?thesis
          by simp
      qed
    qed
  qed
  ultimately show ?thesis
    using enc-upd-def by simp
qed

lemma zip-cont-upd-eq:
  assumes jj < k
    and i = exec t <#> jj
    and i < TT
    and xs ! jj = (0, Some 0)
    and length xs = k
  shows (zip-cont t xs)(i:=enc-upd (zip-cont t xs i) jj (sim-write t ! jj)) =
      zip-cont t (xs[jj:=(1, Some 0)])
    (is ?lhs = ?rhs)
proof
  fix p
  consider p < TT ∧ p ≠ i | p < TT ∧ p = i | p ≥ TT
    by linarith
  then show ?lhs p = ?rhs p
  proof (cases)
    case 1
    then have ?lhs p = zip-cont t xs p
      by simp
    moreover have zip-cont t xs p = zip-cont t (xs[jj:=(1, Some 0)]) p
    proof (rule zip-cont-nth-eqI)
      show p < TT
        using 1 by simp
      show (exec (t + fst (xs ! j)) <:> j) p =
              (exec (t + fst (xs[jj := (1, Some 0)] ! j)) <:> j) p
          if j < k for j
      proof (cases j = jj)
        case True
        then have fst (xs[jj := (1, Some 0)] ! j) = 1
          using assms(1,5) by simp
        then have (exec (t + fst (xs[jj := (1, Some 0)] ! j)) <:> j) p = (exec (Suc t) <:> j) p
          by simp
        also have ... = (exec t <:> j) p
          using assms(2) 1 by (simp add: True assms(1) sim-write)
        finally show ?thesis
          using assms(4) True by simp
      next
        case False
        then show ?thesis
          by simp
      qed
      show snd (xs ! j) = snd (xs[jj := (1, Some 0)] ! j) if j < k for j
        using assms(4,5) that by (cases j = jj) simp-all
```

```
      qed
      ultimately show ?thesis
        by simp
    next
      case 2
      then have ?lhs p = enc-upd (zip-cont t xs i) jj (sim-write t ! jj)
        by simp
      then show ?thesis
        using 2 assms(1,2,4,5) sim-write' zip-cont-nth-eq-updI1 by auto
    next
      case 3
      then show ?thesis
        using zip-cont-def assms(3) by auto
    qed
qed

lemma sem-cmdL5-neq-pos:
  assumes jj < k
    and tps = tpsL t xs i 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and snd (xs ! jj) = Some 0
    and i ≠ exec t <#> jj
    and i < TT
    and tps' = tpsL t xs (Suc i) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows sem (cmdL5 jj) (0, tps) = (0, tps')
proof (rule semI[of 2 * k + 3])
  show proper-command (2 * k + 3) (cmdL5 jj)
    using turing-command-cmdL5[OF assms(1)] turing-commandD(1) by simp
  show length tps = 2 * k + 3
    using assms(2) by simp
  show length tps' = 2 * k + 3
    using assms(6) by simp
  let ?rs = read tps
  have rs1: ?rs ! 1 ≠ □
    using read-tpsL-1-bounds assms(2,5) by (metis not-one-less-zero)
  then show fst (cmdL5 jj ?rs) = 0
    by (simp add: cmdL5-def)
  show act (cmdL5 jj ?rs [!] j) (tps ! j) = tps' ! j if j < 2 * k + 3 for j
  proof -
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
      using ‹j < 2 * k + 3› by linarith
    then show ?thesis
    proof (cases)
      case 1
      then have cmdL5 jj ?rs [!] j = (?rs ! j, Stay)
        using rs1 cmdL5-at-0 by simp
      then show ?thesis
        using assms tpsL-0 1 by (metis act-Stay that length-tpsL)
    next
      case 2
      then have enc-nth (?rs ! 1) (k + jj) = (if i = exec t <#> jj then 1 else 0)
        using assms read-tpsL-1-nth-less-2k by simp
      then have enc-nth (?rs ! 1) (k + jj) = 0
        using assms(4) by simp
      then have ¬ (1 < ?rs ! 1 ∧ ?rs ! 1 < G^(2*k+2)+2 ∧ ?rs ! (3+k+jj) < G ∧ enc-nth (?rs!1) (k+jj) =
1)
        by simp
      then have cmdL5 jj ?rs [!] 1 = (?rs ! 1, Right)
        using cmdL5-at-1-else rs1 by simp
      then show ?thesis
        using assms tpsL-1 2 act-Right that length-tpsL by (metis Suc-eq-plus1 prod.sel(1) tpsL-pos-1)
    next
      case 3
      then have cmdL5 jj ?rs [!] j = (?rs ! j, Stay)
```

```
      using rs1 cmdL5-at-2 by simp
    then show ?thesis
      using assms tpsL-2 3 by (metis act-Stay that length-tpsL)
  next
    case 4
    then have cmdL5 jj ?rs [!] j = (?rs ! j, Stay)
      using rs1 cmdL5-at-3 by simp
    then have act (cmdL5 jj ?rs [!] j) (tps ! j) = tps ! j
      using act-Stay by (simp add: ‹length tps = 2 * k + 3› that)
    then show ?thesis
      using assms tpsL-mvs' by (simp add: 4 add.commute)
  next
    case 5
    then have cmdL5 jj ?rs [!] j = (?rs ! j, Stay)
      using rs1 cmdL5-at-4 by simp
    then have act (cmdL5 jj ?rs [!] j) (tps ! j) = tps ! j
      using act-Stay by (simp add: ‹length tps = 2 * k + 3› that)
    then show ?thesis
      using assms tpsL-symbs' 5 by simp
  qed
 qed
qed

lemma sem-cmdL5-eq-pos:
  assumes jj < k
    and tps = tpsL t xs i 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and xs ! jj = (0, Some 0)
    and i = exec t <#> jj
    and tps' = tpsL t (xs[jj:=(1, Some 0)]) (Suc i) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
  shows sem (cmdL5 jj) (0, tps) = (0, tps')
proof (rule semI[of 2 * k + 3])
  have i < TT
    using exec-pos-less-TT assms(1,4) by simp
  show proper-command (2 * k + 3) (cmdL5 jj)
    using turing-command-cmdL5[OF assms(1)] turing-commandD(1) by simp
  show length tps = 2 * k + 3
    using assms(2) by simp
  show length tps' = 2 * k + 3
    using assms(5) by simp
  let ?rs = read tps
  have rs1: ?rs ! 1 ≠ □
    using read-tpsL-1-bounds assms(2) ‹i < TT› by (metis not-one-less-zero)
  then show fst (cmdL5 jj ?rs) = 0
    by (simp add: cmdL5-def)
  show act (cmdL5 jj (read tps) [!] j) (tps ! j) = tps' ! j if j < 2 * k + 3 for j
  proof -
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
      using ‹j < 2 * k + 3› by linarith
    then show ?thesis
    proof (cases)
      case 1
      then have cmdL5 jj ?rs [!] j = (?rs ! j, Stay)
        using rs1 cmdL5-at-0 by simp
      then show ?thesis
        using assms tpsL-0 1 by (metis act-Stay that length-tpsL)
    next
      case 2
      then have enc-nth (?rs ! 1) (k + jj) = (if i = exec t <#> jj then 1 else 0)
        using assms ‹i < TT› read-tpsL-1-nth-less-2k by simp
      then have enc-nth (?rs ! 1) (k + jj) = 1
        using assms(4) by simp
      moreover have 1 < ?rs ! 1 ∧ ?rs ! 1 < G^(2*k+2)+2
```

using *assms*(*2*) ‹*i* < *TT*› *read-tpsL-1-bounds* **by** *auto*
moreover have *?rs* ! (*3*+*k*+*jj*) < *G*
using *read-tpsL-4 assms sim-write-nth-less-G*[*OF assms*(*1*)] **by** *simp*
ultimately have
1 < *?rs* ! *1* ∧ *?rs* ! *1* < *G*^(*2*∗*k*+*2*)+*2* ∧ *?rs* ! (*3*+*k*+*jj*) < *G* ∧ *enc-nth* (*?rs*!*1*) (*k*+*jj*) = *1*
by *simp*
then have *cmdL5 jj ?rs* [!] *1* = (*enc-upd* (*?rs*!*1*) *jj* (*?rs*!(*3*+*k*+*jj*)), *Right*)
using *cmdL5-at-1 rs1* **by** *simp*
moreover have *?rs*!(*3*+*k*+*jj*) = *sim-write t* ! *jj*
by (*simp add*: *assms*(*1*,*2*) *read-tpsL-4*)
ultimately have ∗: *cmdL5 jj ?rs* [!] *1* = (*enc-upd* (*?rs* ! *1*) *jj* (*sim-write t* ! *jj*), *Right*)
by *simp*

have *?rs* ! *1* = *zip-cont t xs i*
using *assms*(*2*) *read-tpsL-1 zip-cont-def* **by** *auto*

let *?tp* = *act* (*enc-upd* (*?rs* ! *1*) *jj* (*sim-write t* ! *jj*), *Right*) (*tps* ! *j*)
have *?tp* = *tps′* ! *1*
proof −
have *?tp* = ((*zip-cont t xs*)(*i*:=*enc-upd* (*?rs* ! *1*) *jj* (*sim-write t* ! *jj*)), *Suc i*)
using *act-Right′ assms tpsL-1*
by (*metis 2 add.commute fst-conv plus-1-eq-Suc snd-conv*)
moreover have *tps′* ! *1* = (*zip-cont t* (*xs*[*jj*:=(*1*, *Some 0*)]), *Suc i*)
using *assms*(*5*) *tpsL-1* **by** *simp*
moreover have (*zip-cont t xs*)(*i*:=*enc-upd* (*?rs* ! *1*) *jj* (*sim-write t* ! *jj*)) =
zip-cont t (*xs*[*jj*:=(*1*, *Some 0*)])
using *zip-cont-upd-eq assms* ‹*i* < *TT*› ‹*read tps* ! *1* = *zip-cont t xs i*› **by** *auto*
ultimately show *?thesis*
by *auto*
qed
then show *?thesis*
using *2* ∗ **by** *simp*
next
case *3*
then have *cmdL5 jj ?rs* [!] *j* = (*?rs* ! *j*, *Stay*)
using *rs1 cmdL5-at-2* **by** *simp*
then show *?thesis*
using *assms tpsL-2 3* **by** (*metis act-Stay that length-tpsL*)
next
case *4*
then have *cmdL5 jj ?rs* [!] *j* = (*?rs* ! *j*, *Stay*)
using *rs1 cmdL5-at-3* **by** *simp*
then have *act* (*cmdL5 jj ?rs* [!] *j*) (*tps* ! *j*) = *tps* ! *j*
using *act-Stay* **by** (*simp add*: ‹*length tps* = *2* ∗ *k* + *3*› *that*)
then show *?thesis*
using *assms tpsL-mvs′* **by** (*simp add*: *4 add.commute*)
next
case *5*
then have *cmdL5 jj ?rs* [!] *j* = (*?rs* ! *j*, *Stay*)
using *rs1 cmdL5-at-4* **by** *simp*
then have *act* (*cmdL5 jj ?rs* [!] *j*) (*tps* ! *j*) = *tps* ! *j*
using *act-Stay* **by** (*simp add*: ‹*length tps* = *2* ∗ *k* + *3*› *that*)
then show *?thesis*
using *assms tpsL-symbs′ 5* **by** *simp*
qed
qed
qed

lemma *sem-cmdL5-eq-TT*:
assumes *jj* < *k* **and** *tps* = *tpsL t xs TT q mvs symbs*
shows *sem* (*cmdL5 jj*) (*0*, *tps*) = (*1*, *tps*)
proof (*rule semI*[*of 2* ∗ *k* + *3*])
show *proper-command* (*2* ∗ *k* + *3*) (*cmdL5 jj*)

```
    using turing-command-cmdL5[OF assms(1)] turing-commandD(1) by simp
  show length tps = 2 * k + 3
    using assms(2) by simp
  show len: length tps = 2 * k + 3
    using assms(2) by simp
  let ?rs = read tps
  have rs1: ?rs ! 1 = □
    using read-tpsL-1 assms(2) by simp
  then show fst (cmdL5 jj ?rs) = 1
    by (simp add: cmdL5-def)
  show ⋀i. i < 2 * k + 3 ⟹ act (cmdL5 jj ?rs [!] i) (tps ! i) = tps ! i
    using len rs1 act-Stay cmdL5-eq-0 read-length by auto
qed

lemma execute-tmL45-1:
  assumes tt ≤ exec t <#> jj
    and jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and xs ! jj = (0, Some 0)
    and tps' = tpsL t xs tt 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows execute (tmL45 jj) (0, tps) tt = (0, tps')
  using assms(1,5)
proof (induction tt arbitrary: tps')
  case 0
  then show ?case
    using assms(2−4) by simp
next
  case (Suc tt)
  then have tt-neq: tt ≠ exec t <#> jj
    by simp
  have tt-le: tt ≤ exec t <#> jj
    using Suc.prems by simp
  then have tt-less: tt < TT
    using exec-pos-less-TT assms(2) by (meson le-trans less-Suc-eq-le)
  define tps-tt where tps-tt = tpsL t xs tt 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  have execute (tmL45 jj) (0, tps) (Suc tt) = exe (tmL45 jj) (execute (tmL45 jj) (0, tps) tt)
    by simp
  also have ... = exe (tmL45 jj) (0, tps-tt)
    using Suc.IH assms(2−4) tt-le tps-tt-def by simp
  also have ... = sem (cmdL5 jj) (0, tps-tt)
    using tmL45-def exe-lt-length by simp
  also have ... = (0, tpsL t xs (Suc tt) 1 (λj. sim-move t ! j) (λj. sim-write t ! j))
    using sem-cmdL5-neq-pos assms(2−4) tt-neq tt-less by (simp add: tps-tt-def)
  finally show ?case
    by (simp add: Suc.prems(2))
qed

lemma execute-tmL45-2:
  assumes tt = exec t <#> jj
    and jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and xs ! jj = (0, Some 0)
    and tps' = tpsL t (xs[jj:=(1, Some 0)]) (Suc tt) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
  shows execute (tmL45 jj) (0, tps) (Suc tt) = (0, tps')
proof −
  have execute (tmL45 jj) (0, tps) (Suc tt) = exe (tmL45 jj) (execute (tmL45 jj) (0, tps) tt)
    by simp
  also have ... = exe (tmL45 jj) (0, tpsL t xs tt 1 (λj. sim-move t ! j) (λj. sim-write t ! j))
    using execute-tmL45-1 assms by simp
  also have ... = sem (cmdL5 jj) (0, tpsL t xs tt 1 (λj. sim-move t ! j) (λj. sim-write t ! j))
    using tmL45-def exe-lt-length by simp
  also have ... = (0, tpsL t (xs[jj:=(1, Some 0)]) (Suc tt) 1 (λj. sim-move t ! j) (λj. sim-write t ! j))
```

using *sem-cmdL5-eq-pos*[*OF assms*(*2*)] *assms* **by** *simp*
finally show *?thesis*
using *assms*(*5*) **by** *simp*
qed

lemma *execute-tmL45-3*:
assumes *tt* ≥ *Suc* (*exec t* <#> *jj*)
and *tt* ≤ *TT*
and *jj* < *k*
and *tps* = *tpsL t xs 0 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
and *xs* ! *jj* = (*0*, *Some 0*)
and *tps'* = *tpsL t* (*xs*[*jj*:=(*1*, *Some 0*)]) *tt 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
and *length xs* = *k*
shows *execute* (*tmL45 jj*) (*0*, *tps*) *tt* = (*0*, *tps'*)
using *assms*(*1*,*2*,*6*)
proof (*induction tt arbitrary*: *tps' rule*: *nat-induct-at-least*)
case *base*
then show *?case*
using *assms*(*3*−*5*,*7*) *execute-tmL45-2* **by** *simp*
next
case (*Suc tt*)
then have *tt*: *tt* < *TT tt* ≠ *exec t* <#> *jj*
by *simp-all*
have *execute* (*tmL45 jj*) (*0*, *tps*) (*Suc tt*) = *exe* (*tmL45 jj*) (*execute* (*tmL45 jj*) (*0*, *tps*) *tt*)
by *simp*
also have ... = *exe* (*tmL45 jj*) (*0*, *tpsL t* (*xs*[*jj*:=(*1*, *Some 0*)]) *tt 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*))
using *Suc* **by** *simp*
also have ... = *sem* (*cmdL5 jj*) (*0*, *tpsL t* (*xs*[*jj*:=(*1*, *Some 0*)]) *tt 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*))
using *tmL45-def exe-lt-length* **by** *simp*
also have ... = (*0*, *tpsL t* (*xs*[*jj*:=(*1*, *Some 0*)]) (*Suc tt*) *1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*))
using *sem-cmdL5-neq-pos tt* **by** (*simp add*: *assms*(*3*) *assms*(*7*))
finally show *?case*
using *Suc*(*4*) **by** *presburger*
qed

lemma *execute-tmL45-4*:
assumes *jj* < *k*
and *tps* = *tpsL t xs 0 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
and *xs* ! *jj* = (*0*, *Some 0*)
and *tps'* = *tpsL t* (*xs*[*jj*:=(*1*, *Some 0*)]) *TT 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
and *length xs* = *k*
shows *execute* (*tmL45 jj*) (*0*, *tps*) (*Suc TT*) = (*1*, *tps'*)
proof −
have *execute* (*tmL45 jj*) (*0*, *tps*) (*Suc TT*) = *exe* (*tmL45 jj*) (*execute* (*tmL45 jj*) (*0*, *tps*) *TT*)
by *simp*
also have ... = *exe* (*tmL45 jj*) (*0*, *tps'*)
using *assms execute-tmL45-3* **by** (*metis Suc-leI exec-pos-less-TT order-refl*)
also have ... = *sem* (*cmdL5 jj*) (*0*, *tps'*)
using *tmL45-def exe-lt-length* **by** *simp*
also have ... = (*1*, *tps'*)
using *sem-cmdL5-eq-TT assms*(*1*,*4*) **by** *simp*
finally show *?thesis* **.**
qed

definition *esL45* ≡ *map* (λ*i*. (*n* + *1*, *Suc i*)) [*0*..<*TT*] @ [(*n* + *1*, *TT*)]

lemma *len-esL45*: *length esL45* = *Suc TT*
using *esL45-def* **by** *simp*

lemma *nth-map-upt-TT*:
fixes *es*
assumes *es* = *map f* [*0*..<*TT*] @ *es'* **and** *i* < *TT*
shows *es* ! *i* = *f i*

```
  using assms by (metis add.left-neutral diff-zero length-map length-upt nth-append nth-map nth-upt)

lemma tmL45:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (0, Some 0)
    and tps' = tpsL t (xs[jj:=(1, Some 0)]) TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL45 jj) tps esL45 tps'
proof
  show execute (tmL45 jj) (0, tps) (length esL45) = (length (tmL45 jj), tps')
    using tmL45-def execute-tmL45-4 esL45-def assms by simp
  show fst (execute (tmL45 jj) (0, tps) i) < length (tmL45 jj) if i < length esL45 for i
  proof -
    have i ≤ TT
      using esL45-def that by simp
    then consider i ≤ exec t <#> jj | i = Suc (exec t <#> jj) | i > Suc (exec t <#> jj) ∧ i ≤ TT
      by linarith
    then show ?thesis
    proof (cases)
      case 1
      then show ?thesis
        using assms execute-tmL45-1 tmL45-def by simp
    next
      case 2
      then show ?thesis
        using assms execute-tmL45-2 tmL45-def by simp
    next
      case 3
      then show ?thesis
        using assms execute-tmL45-3 tmL45-def by simp
    qed
  qed
  show execute (tmL45 jj) (0, tps) (Suc i) <#> 0 = fst (esL45 ! i) ∧
      execute (tmL45 jj) (0, tps) (Suc i) <#> 1 = snd (esL45 ! i)
    if i < length esL45 for i
  proof -
    have i ≤ TT
      using esL45-def that by simp
    then consider i < exec t <#> jj | i = exec t <#> jj | i ≥ Suc (exec t <#> jj) ∧ i < TT | i = TT
      by linarith
    then show ?thesis
    proof (cases)
      case 1
      then have Suc i ≤ exec t <#> jj
        by simp
      then have i < TT
        using exec-pos-less-TT by (metis ‹i ≤ TT› assms(1) nat-less-le not-less-eq-eq)
      then have esL45 ! i = (n + 1, Suc i)
        using esL45-def nth-map-upt-TT by auto
      then show ?thesis
       using assms execute-tmL45-1 tpsL-pos-0 tpsL-pos-1 by (metis ‹Suc i ≤ exec t <#> jj› fst-conv snd-conv)
    next
      case 2
      then have i < TT
        using exec-pos-less-TT by (simp add: assms(1))
      then have esL45 ! i = (n + 1, Suc i)
        using esL45-def nth-map-upt-TT by auto
      then show ?thesis
        using assms execute-tmL45-2 tpsL-pos-0 tpsL-pos-1 by (metis 2 fst-conv snd-conv)
    next
      case 3
      then have esL45 ! i = (n + 1, Suc i)
```

```
      using esL45-def nth-map-upt-TT by auto
    then show ?thesis
      using assms execute-tmL45-3 tpsL-pos-0 tpsL-pos-1 3
      by (metis Suc-leI fst-conv le-imp-less-Suc nat-less-le snd-conv)
  next
    case 4
    then have esL45 ! i = (n + 1, TT)
      using esL45-def by (simp add: nth-append)
    then show ?thesis
      using assms execute-tmL45-4 tpsL-pos-0 tpsL-pos-1 4 by simp
  qed
 qed
qed

definition esL46 ≡ esL45 @ [(n + 1, fmt n)]

lemma len-esL46: length esL46 = TT + 2
 using len-esL45 esL46-def by simp

lemma tmL46:
 assumes jj < k
  and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  and length xs = k
  and xs ! jj = (0, Some 0)
  and tps' = tpsL t (xs[jj:=(1, Some 0)]) (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
 shows traces (tmL46 jj) tps esL46 tps'
 unfolding tmL46-def esL46-def
proof (rule traces-sequential)
 let ?tps = tpsL t (xs[jj:=(1, Some 0)]) TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
 show traces (tmL45 jj) tps esL45 ?tps
  using tmL45 assms by simp
 show traces (tm-left 1) ?tps [(n + 1, fmt n)] tps'
  using tpsL-pos-0 tpsL-pos-1 assms tpsL-def tpsL-1
  by (intro traces-tm-left-1I) simp-all
qed

lemma sem-cmdL7-nonleft-gt-0:
 assumes jj < k
  and tps = tpsL t xs i 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  and length xs = k
  and i < TT
  and i > 0
  and sim-move t ! jj ≠ 0
  and tps' = tpsL t xs (i − 1) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
 shows sem (cmdL7 jj) (0, tps) = (0, tps')
proof (rule semI[of 2 ∗ k + 3])
 show proper-command (2 ∗ k + 3) (cmdL7 jj)
  using turing-command-cmdL7[OF assms(1)] turing-commandD(1) by simp
 show len: length tps = 2 ∗ k + 3
  using assms(2) by simp
 show length tps' = 2 ∗ k + 3
  using assms(7) by simp
 define rs where rs = read tps
 then have ¬ is-beginning (rs ! 1)
  using read-tpsL-1-nth-2k1 assms
  by (metis enc-nth-dec nat-neq-iff numerals(1) zero-neq-numeral)
 then show fst (cmdL7 jj rs) = 0
  unfolding cmdL7-def by simp

 have rs ! (3 + jj) = sim-move t ! jj
  using rs-def assms(1,2) read-tpsL-3 by simp
 moreover have sim-move t ! jj < 3
  using sim-move-def assms(1) direction-to-symbol-less sim-move-nth sim-move-nth-else
```

```
    by (metis One-nat-def not-add-less2 not-less-eq numeral-3-eq-3 plus-1-eq-Suc)
  ultimately have condition7c rs jj
    using assms(6) by simp
  then have *: snd (cmdL7 jj rs) =
    [(rs ! 0, Stay),
     (rs ! 1, Left),
     (rs ! 2, Stay)] @
     (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
     (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
    unfolding cmdL7-def by auto

  show act (cmdL7 jj (read tps) [!] j) (tps ! j) = tps' ! j if j < 2 ∗ k + 3 for j
  proof −
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 ∗ k + 3
      using ‹j < 2 ∗ k + 3› by linarith
    then show ?thesis
    proof (cases)
      case 1
      then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 0) (tps ! 0)
        by simp
      also have ... = act (rs ! 0, Stay) (tps ! 0)
        using ∗ rs-def by simp
      also have ... = tps ! 0
        using act-Stay len rs-def by simp
      also have ... = tps' ! 0
        using assms(2,7) tpsL-0 by simp
      also have ... = tps' ! j
        using 1 by simp
      finally show ?thesis .
    next
      case 2
      then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 1) (tps ! 1)
        by simp
      also have ... = act (rs ! 1, Left) (tps ! 1)
        using ∗ rs-def by simp
      also have ... = tps' ! 1
        using act-Left len rs-def assms tpsL-1 by (metis 2 fst-conv that tpsL-pos-1)
      also have ... = tps' ! j
        using 2 by simp
      finally show ?thesis .
    next
      case 3
      then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 2) (tps ! 2)
        by simp
      also have ... = act (rs ! 2, Stay) (tps ! 2)
        using ∗ rs-def by simp
      also have ... = tps ! 2
        using act-Stay len rs-def by simp
      also have ... = tps' ! 2
        using assms(2,7) tpsL-2 by simp
      also have ... = tps' ! j
        using 3 by simp
      finally show ?thesis .
    next
      case 4
      then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
        using ∗ rs-def threeplus2k-2[where ?a=(rs ! 0, Stay)] by simp
      also have ... = tps ! j
        using len act-Stay rs-def that by simp
      also have ... = tps' ! j
        using assms(2,7) tpsL-mvs' 4 by simp
      finally show ?thesis .
    next
```

case *5*
then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*rs* ! *j*, *Stay*) (*tps* ! *j*)
using ∗ *rs-def threeplus2k-3*[where *?a*=(*rs* ! *0*, *Stay*)] by *simp*
also have ... = *tps* ! *j*
using *len act-Stay rs-def that* by *simp*
also have ... = *tps′* ! *j*
using *assms*(*2*,*7*) *tpsL-symbs′ 5* by *simp*
finally show *?thesis* .
qed
qed
qed

lemma *sem-cmdL7-nonleft-eq-0*:
assumes *jj* < *k*
and *tps* = *tpsL t xs 0 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
and *length xs* = *k*
and *sim-move t* ! *jj* ≠ *0*
shows *sem* (*cmdL7 jj*) (*0*, *tps*) = (*1*, *tps*)
proof (*rule semI*[*of 2* ∗ *k* + *3*])
show *proper-command* (*2* ∗ *k* + *3*) (*cmdL7 jj*)
using *turing-command-cmdL7*[*OF assms*(*1*)] *turing-commandD*(*1*) by *simp*
show *len*: *length tps* = *2* ∗ *k* + *3*
using *assms*(*2*) by *simp*
show *length tps* = *2* ∗ *k* + *3*
using *assms*(*2*) by *simp*

define *rs* where *rs* = *read tps*
then have *is-beginning* (*rs* ! *1*)
using *read-tpsL-1-nth-2k1 assms enc-nth-def read-tpsL-1-bounds zero-less-Suc*
by *simp*
then show *fst* (*cmdL7 jj* (*read tps*)) = *1*
using *cmdL7-def rs-def* by *simp*

have *rs* ! (*3* + *jj*) = *sim-move t* ! *jj*
using *rs-def assms*(*1*,*2*) *read-tpsL-3* by *simp*
then have *condition7c rs jj*
using *sim-move direction-to-symbol-less sim-move-nth sim-move-nth-else assms*(*1*,*4*)
by (*metis less-Suc-eq not-add-less2 numeral-3-eq-3 numeral-eq-iff numerals*(*1*) *plus-1-eq-Suc semiring-norm*(*86*))
then have ∗: *snd* (*cmdL7 jj rs*) =
[(*rs* ! *0*, *Stay*),
(*rs* ! *1*, *Left*),
(*rs* ! *2*, *Stay*)] @
(*map* (λ*j*. (*rs* ! (*j* + *3*), *Stay*)) [*0*..<*k*]) @
(*map* (λ*j*. (*rs* ! (*3* + *k* + *j*), *Stay*)) [*0*..<*k*])
unfolding *cmdL7-def* by *auto*

show *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *tps* ! *j* if *j* < *2* ∗ *k* + *3* for *j*
proof −
consider *j* = *0* | *j* = *1* | *j* = *2* | *3* ≤ *j* ∧ *j* < *k* + *3* | *k* + *3* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
using ‹*j* < *2* ∗ *k* + *3*› by *linarith*
then show *?thesis*
proof (*cases*)
case *1*
then show *?thesis*
using ∗ *act-Stay assms*(*2*) *rs-def* by *simp*
next
case *2*
then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*rs* ! *1*, *Left*) (*tps* ! *j*)
using ∗ *rs-def* by *simp*
also have ... = *tps* ! *j*
using *2 assms*(*2*) *act-Left that length-tpsL tpsL-1 tpsL-pos-1 rs-def*
by (*metis diff-is-0-eq′ fst-conv less-numeral-extra*(*1*) *nat-less-le*)
finally show *?thesis*

    by *simp*
  next
    case *3*
    then show *?thesis*
      using ∗ *act-Stay assms(2) rs-def* **by** *simp*
  next
    case *4*
    then show *?thesis*
      using ∗ *act-Stay assms rs-def threeplus2k-2*[**where** *?a=(rs ! 0, Stay)*] *len* **by** *simp*
  next
    case *5*
    then show *?thesis*
      using ∗ *act-Stay assms rs-def threeplus2k-3*[**where** *?a=(rs ! 0, Stay)*] *len* **by** *simp*
  qed
 qed
qed

lemma *execute-tmL67-nonleft-less*:
 assumes *jj < k*
  and *tps = tpsL t xs (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)*
  and *length xs = k*
  and *sim-move t ! jj ≠ 0*
  and *tt < TT*
  and *tps′ = tpsL t xs (fmt n − tt) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)*
 shows *execute (tmL67 jj) (0, tps) tt = (0, tps′)*
 using *assms(5,6)*
proof (*induction tt arbitrary: tps′*)
 case *0*
 then show *?case*
  using *assms(1−4) tmL67-def* **by** *simp*
next
 case (*Suc tt*)
 have *execute (tmL67 jj) (0, tps) (Suc tt) = exe (tmL67 jj) (execute (tmL67 jj) (0, tps) tt)*
  by *simp*
 also have ... = *exe (tmL67 jj) (0, tpsL t xs (fmt n − tt) 1 (λj. sim-move t ! j) (λj. sim-write t ! j))*
  using *Suc* **by** *simp*
 finally show *?case*
  using *assms(1−4) sem-cmdL7-nonleft-gt-0 tmL67-def exe-lt-length Suc* **by** *simp*
qed

lemma *execute-tmL67-nonleft-finish*:
 assumes *jj < k*
  and *tps = tpsL t xs (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)*
  and *length xs = k*
  and *sim-move t ! jj ≠ 0*
  and *tps′ = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)*
 shows *execute (tmL67 jj) (0, tps) TT = (1, tps′)*
 using *assms execute-tmL67-nonleft-less sem-cmdL7-nonleft-eq-0 tmL67-def exe-lt-length*
 by *simp*

definition *esL67 ≡ map (λi. (n + 1, i)) (rev [0..<fmt n]) @ [(n + 1, 0)]*

lemma *esL67-at-fmtn* [*simp*]: *esL67 ! (fmt n) = (n + 1, 0)*
 using *esL67-def* **by** (*simp add: nth-append*)

lemma *esL67-at-lt-fmtn* [*simp*]: *i < fmt n ⟹ esL67 ! i = (n + 1, fmt n − i − 1)*
proof −
 assume *i < fmt n*
 then have *(rev [0..<fmt n]) ! i = fmt n − 1 − i*
  by (*metis Suc-diff-1 add.left-neutral bot-nat-0.extremum diff-diff-add diff-less-Suc diff-zero length-upt less-nat-zero-code nat-less-le nth-upt plus-1-eq-Suc rev-nth*)
 moreover have *length (map (Pair (Suc n)) (rev [0..<fmt n])) = fmt n*
  by *simp*

```
  ultimately show ?thesis
    by (simp add: ‹i < fmt n› esL67-def nth-append)
qed

lemma tmL67-nonleft:
  assumes jj < k
    and tps = tpsL t xs (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and sim-move t ! jj ≠ 0
    and tps' = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL67 jj) tps esL67 tps'
proof
  have len: length esL67 = TT
    using esL67-def by simp
  then show 1: execute (tmL67 jj) (0, tps) (length esL67) = (length (tmL67 jj), tps')
    using assms tmL67-def execute-tmL67-nonleft-finish by simp
  show ⋀i. i < length esL67 ⟹
        fst (execute (tmL67 jj) (0, tps) i) < length (tmL67 jj)
    using len assms execute-tmL67-nonleft-less tmL67-def by simp
  show (execute (tmL67 jj) (0, tps) (Suc i)) <#> 0 = fst (esL67 ! i) ∧
        (execute (tmL67 jj) (0, tps) (Suc i)) <#> 1 = snd (esL67 ! i)
      if i < length esL67 for i
  proof (cases i = fmt n)
    case True
    then show ?thesis
      using assms that 1 tpsL-pos-0 tpsL-pos-1 len by simp
  next
    case False
    then have Suc i < TT
      using that len by simp
    moreover from this have esL67 ! i = (n + 1, fmt n − 1 − i)
      by simp
    ultimately show ?thesis
      using tpsL-pos-0 tpsL-pos-1 assms(1−5) execute-tmL67-nonleft-less
      by (metis (no-types, lifting) diff-diff-left fst-conv plus-1-eq-Suc snd-conv)
  qed
qed

lemma sem-cmdL7-1:
  assumes jj < k
    and tps = tpsL t xs i 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (1, Some 0)
    and i < TT
    and i > exec t <#> jj
    and sim-move t ! jj = 0
    and tps' = tpsL t xs (i − 1) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows sem (cmdL7 jj) (0, tps) = (0, tps')
proof (rule semI[of 2 ∗ k + 3])
  show proper-command (2 ∗ k + 3) (cmdL7 jj)
    using turing-command-cmdL7[OF assms(1)] turing-commandD(1) by simp
  show len: length tps = 2 ∗ k + 3
    using assms(2) by simp
  show length tps' = 2 ∗ k + 3
    using assms(8) by simp

  define rs where rs = read tps
  then have not-beginning: ¬ is-beginning (rs ! 1)
    using read-tpsL-1-nth-2k1 assms enc-nth-def read-tpsL-1-bounds zero-less-Suc
    by simp
  then show fst (cmdL7 jj (read tps)) = 0
    using cmdL7-def rs-def by simp
```

```
have rs ! (3 + jj) = □
  using rs-def read-tpsL-3 assms by simp
moreover have enc-nth (rs ! 1) (k + jj) = 0
  using assms rs-def read-tpsL-1-nth-less-2k by simp
ultimately have condition7c rs jj
  using not-beginning by simp
then have *: snd (cmdL7 jj rs) =
  [(rs ! 0, Stay),
   (rs ! 1, Left),
   (rs ! 2, Stay)] @
   (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
   (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
  unfolding cmdL7-def by auto

show act (cmdL7 jj (read tps) [!] j) (tps ! j) = tps' ! j if j < 2 * k + 3 for j
proof −
  consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
    using ‹j < 2 * k + 3› by linarith
  then show ?thesis
  proof (cases)
    case 1
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 0) (tps ! 0)
      by simp
    also have ... = act (rs ! 0, Stay) (tps ! 0)
      using * rs-def by simp
    also have ... = tps ! 0
      using act-Stay len rs-def by simp
    also have ... = tps' ! 0
      using assms(2,8) tpsL-0 by simp
    also have ... = tps' ! j
      using 1 by simp
    finally show ?thesis .
  next
    case 2
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 1) (tps ! 1)
      by simp
    also have ... = act (rs ! 1, Left) (tps ! 1)
      using * rs-def by simp
    also have ... = tps' ! 1
      using act-Left len rs-def assms tpsL-1 by (metis 2 fst-conv that tpsL-pos-1)
    also have ... = tps' ! j
      using 2 by simp
    finally show ?thesis .
  next
    case 3
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 2) (tps ! 2)
      by simp
    also have ... = act (rs ! 2, Stay) (tps ! 2)
      using * rs-def by simp
    also have ... = tps ! 2
      using act-Stay len rs-def by simp
    also have ... = tps' ! 2
      using assms(2,8) tpsL-2 by simp
    also have ... = tps' ! j
      using 3 by simp
    finally show ?thesis .
  next
    case 4
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
      using * rs-def threeplus2k-2[where ?a=(rs ! 0, Stay)] by simp
    also have ... = tps ! j
      using len act-Stay rs-def that by simp
    also have ... = tps' ! j
```

```
      using assms(2,8) tpsL-mvs' 4 by simp
    finally show ?thesis .
  next
    case 5
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
      using ∗ rs-def threeplus2k-3[where ?a=(rs ! 0, Stay)] by simp
    also have ... = tps ! j
      using len act-Stay rs-def that by simp
    also have ... = tps' ! j
      using assms(2,8) tpsL-symbs' 5 by simp
    finally show ?thesis .
  qed
 qed
qed

lemma execute-tmL67-1:
 assumes jj < k
  and tps = tpsL t xs (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  and length xs = k
  and xs ! jj = (1, Some 0)
  and sim-move t ! jj = 0
  and tt < TT − exec t <#> jj
  and tps' = tpsL t xs (fmt n − tt) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
 shows execute (tmL67 jj) (0, tps) tt = (0, tps')
 using assms(6,7)
proof (induction tt arbitrary: tps')
 case 0
 then show ?case
  by (simp add: assms(2))
next
 case (Suc tt)
 then have execute (tmL67 jj) (0, tps) (Suc tt) = sem (cmdL7 jj) (execute (tmL67 jj) (0, tps) tt)
  using exe-lt-length tmL67-def by simp
 then show ?case
  using assms(1−5) sem-cmdL7-1 Suc by simp
qed

lemma zip-cont-enc-upd-Some:
 assumes jj < k
  and length xs = k
  and xs ! jj = (1, None)
  and i = exec (Suc t) <#> jj
 shows (zip-cont t xs)(i:=(enc-upd (zip-cont t xs i) (k + jj) 1)) = zip-cont t (xs[jj:=(1, Some 1)])
  (is ?lhs = ?rhs)
proof
 fix p
 have i < TT
  using assms(1,4) exec-pos-less-TT by simp

 consider p < TT ∧ p ≠ i | p < TT ∧ p = i | p ≥ TT
  by linarith
 then show ?lhs p = ?rhs p
 proof (cases)
  case 1
  then have ?lhs p = zip-cont t xs p
   by simp
  moreover have zip-cont t xs p = zip-cont t (xs[jj:=(1, Some 1)]) p
  proof (rule zip-cont-eqI)
   show p < TT
    using 1 by simp
   show (exec (t + fst (xs ! j)) <:> j) p = (exec (t + fst (xs[jj := (1, Some 1)] ! j)) <:> j) p
    if j < k for j
    using assms(1−3) by (cases j = jj) simp-all
```

```
      show (case snd (xs ! j) of None ⇒ 0 | Some d ⇒ if p = (exec (t + d)) <#> j then 1 else 0) =
        (case snd (xs[jj := (1, Some 1)] ! j) of None ⇒ 0 | Some d ⇒ if p = (exec (t + d)) <#> j then 1 else 0)
          (is ?lhs = ?rhs)
        if j < k for j
      proof (cases j = jj)
        case True
        then show ?thesis
          using 1 assms by simp
      next
        case False
        then show ?thesis
          by simp
      qed
    qed
    ultimately show ?thesis
      by simp
  next
    case 2
    then show ?thesis
      using assms enc-upd-zip-cont-None-Some by simp
  next
    case 3
    then show ?thesis
      using ‹i < TT› zip-cont-eq-0 by simp
  qed
qed

lemma zip-cont-enc-upd-Some-Right:
  assumes jj < k
    and length xs = k
    and xs ! jj = (1, None)
    and i = Suc (exec t <#> jj)
    and sim-move t ! jj = 2
  shows (zip-cont t xs)(i:=(enc-upd (zip-cont t xs i) (k + jj) 1)) = zip-cont t (xs[jj:=(1, Some 1)])
proof −
  have i = exec (Suc t) <#> jj
    using assms sim-move by simp
  then show ?thesis
    using zip-cont-enc-upd-Some[OF assms(1−3)] by simp
qed

lemma zip-cont-enc-upd-Some-Left:
  assumes jj < k
    and length xs = k
    and xs ! jj = (1, None)
    and Suc i = exec t <#> jj
    and sim-move t ! jj = 0
  shows (zip-cont t xs)(i:=(enc-upd (zip-cont t xs i) (k + jj) 1)) = zip-cont t (xs[jj:=(1, Some 1)])
    (is ?lhs = ?rhs)
proof −
  have i = exec (Suc t) <#> jj
    using assms sim-move by simp
  then show ?thesis
    using zip-cont-enc-upd-Some[OF assms(1−3)] by simp
qed

lemma zip-cont-enc-upd-None:
  assumes jj < k
    and length xs = k
    and xs ! jj = (1, Some 0)
    and i = exec t <#> jj
  shows (zip-cont t xs)(i:=(enc-upd (zip-cont t xs i) (k + jj) 0)) = zip-cont t (xs[jj:=(1, None)])
    (is ?lhs = ?rhs)
```

proof
  fix *p*
  consider $p < TT \wedge p \neq i \mid p < TT \wedge p = i \mid p \geq TT$
    by *linarith*
  then show *?lhs p = ?rhs p*
  proof (*cases*)
    case *1*
    then have *?lhs p = zip-cont t xs p*
      by *simp*
    moreover have *zip-cont t xs p = zip-cont t (xs[jj:=(1, None)]) p*
    proof (*rule zip-cont-eqI*)
      show $p < TT$
        using *1* **by** *simp*
      show *(exec (t + fst (xs ! j)) <:> j) p = (exec (t + fst (xs[jj := (1, None)] ! j)) <:> j) p*
        if $j < k$ **for** *j*
        using *assms(1−3)* **by** *(cases j = jj) simp-all*
      show *(case snd (xs ! j) of None ⇒ 0 | Some d ⇒ if p = (exec (t + d)) <#> j then 1 else 0) =*
          (case snd (xs[jj := (1, None)] ! j) of None ⇒ 0 | Some d ⇒ if p = (exec (t + d)) <#> j then 1 else 0)
        if $j < k$ **for** *j*
      using *assms 1* **by** *(cases j = jj) simp-all*
    qed
    ultimately show *?thesis*
      by *simp*
  next
    case *2*
    then have *?lhs p = enc-upd (zip-cont t xs i) (k + jj) 0*
      by *simp*
    moreover have *enc-upd (zip-cont t xs i) (k + jj) 0 = zip-cont t (xs[jj:=(1, None)]) i*
      using *assms(1−4) enc-upd-zip-cont-Some-None* **by** *simp*
    ultimately show *?thesis*
      using *2* **by** *simp*
  next
    case *3*
    moreover have $i < TT$
      using *assms(4)* **by** *(simp add: assms(1) exec-pos-less-TT)*
    ultimately show *?thesis*
      using *zip-cont-eq-0* **by** *simp*
  qed
qed

lemma *sem-cmdL7-2a*:
  assumes $jj < k$
    and *tps = tpsL t xs i 1 (λj. sim-move t ! j) (λj. sim-write t ! j)*
    and *length xs = k*
    and *xs ! jj = (1, Some 0)*
    and *i = exec t <#> jj*
    and $i > 0$
    and *sim-move t ! jj = 0*
    and *tps' = tpsL t (xs[jj:=(1, None)]) (i − 1) 1 (λj. if j = jj then 3 else sim-move t ! j) (λj. sim-write t ! j)*
  shows *sem (cmdL7 jj) (0, tps) = (0, tps')*
proof (*rule semI[of 2 ∗ k + 3]*)
  show *proper-command (2 ∗ k + 3) (cmdL7 jj)*
    using *turing-command-cmdL7[OF assms(1)] turing-commandD(1)* **by** *simp*
  show *len*: *length tps = 2 ∗ k + 3*
    using *assms(2)* **by** *simp*
  show *length tps' = 2 ∗ k + 3*
    using *assms(8)* **by** *simp*

  define *rs* **where** *rs = read tps*
  then have *not-beginning*: *¬ is-beginning (rs ! 1)*
    using *read-tpsL-1-nth-2k1 assms enc-nth-def read-tpsL-1-bounds zero-less-Suc exec-pos-less-TT*
    by *simp*
  then show *fst (cmdL7 jj (read tps)) = 0*

```
  using cmdL7-def rs-def by simp

have i < TT
  using assms(5) by (simp add: assms(1) exec-pos-less-TT)

have rs ! (3 + jj) = □
  using rs-def read-tpsL-3 assms by simp
moreover have enc-nth (rs ! 1) (k + jj) = 1
  using assms rs-def read-tpsL-1-nth-less-2k[OF ‹i < TT›] by simp
ultimately have condition7a rs jj
  using not-beginning ‹i < TT› assms(2) read-tpsL-1-bounds rs-def by simp
then have *: snd (cmdL7 jj rs) =
  [(rs ! 0, Stay),
   (enc-upd (rs ! 1) (k + jj) 0, Left),
   (rs ! 2, Stay)] @
   (map (λj. (if j = jj then 3 else rs ! (j + 3), Stay)) [0..<k]) @
   (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
  unfolding cmdL7-def by auto

show act (cmdL7 jj (read tps) [!] j) (tps ! j) = tps' ! j if j < 2 * k + 3 for j
proof –
  consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
    using ‹j < 2 * k + 3› by linarith
  then show ?thesis
  proof (cases)
    case 1
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 0) (tps ! 0)
      by simp
    also have ... = act (rs ! 0, Stay) (tps ! 0)
      using * rs-def by simp
    also have ... = tps ! 0
      using act-Stay len rs-def by simp
    also have ... = tps' ! 0
      using assms(2,8) tpsL-0 by simp
    also have ... = tps' ! j
      using 1 by simp
    finally show ?thesis .
  next
    case 2
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 1) (tps ! 1)
      by simp
    also have ... = act (enc-upd (rs ! 1) (k + jj) 0, Left) (tps ! 1)
      using * rs-def by simp
    also have ... = tps ! 1 |:=| (enc-upd (rs ! 1) (k + jj) 0) |–| 1
      using act-Left' 2 len by simp
    also have ... = tps' ! 1
      using assms zip-cont-enc-upd-None rs-def read-tpsL-1 tpsL-1 zip-cont-def by simp
    finally show ?thesis
      using 2 by simp
  next
    case 3
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 2) (tps ! 2)
      by simp
    also have ... = act (rs ! 2, Stay) (tps ! 2)
      using * rs-def by simp
    also have ... = tps ! 2
      using act-Stay len rs-def by simp
    also have ... = tps' ! 2
      using assms(2,8) tpsL-2 by simp
    also have ... = tps' ! j
      using 3 by simp
    finally show ?thesis .
  next
```

```
      case 4
      show ?thesis
      proof (cases j = 3 + jj)
        case True
        then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (3, Stay) (tps ! j)
          using ∗ rs-def threeplus2k-2[where ?a=(rs ! 0, Stay)] 4 diff-add-inverse by (smt (verit))
        also have ... = tps′ ! j
          using 4 assms(2,8) True act-onesie tpsL-mvs by simp
        finally show ?thesis .
      next
        case False
        then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
          using ∗ rs-def threeplus2k-2[where ?a=(rs ! 0, Stay)] 4 diff-add-inverse by auto
        also have ... = tps′ ! j
          using 4 assms(2,8) False act-Stay len rs-def that tpsL-mvs′
          by (smt (verit) add.commute le-add-diff-inverse2)
        finally show ?thesis .
      qed
    next
      case 5
      then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
        using ∗ rs-def threeplus2k-3[where ?a=(rs ! 0, Stay)] by simp
      also have ... = tps ! j
        using len act-Stay rs-def that by simp
      also have ... = tps′ ! j
        using assms(2,8) tpsL-symbs′ 5 by simp
      finally show ?thesis .
    qed
  qed
qed

lemma execute-tmL67-2a:
  assumes jj < k
    and tps = tpsL t xs (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (1, Some 0)
    and sim-move t ! jj = 0
    and exec t <#> jj > 0
    and tt = TT − exec t <#> jj
    and tps′ = tpsL t (xs[jj:=(1, None)]) (fmt n − tt) 1 (λj. if j = jj then 3 else sim-move t ! j) (λj. sim-write
t ! j)
  shows execute (tmL67 jj) (0, tps) tt = (0, tps′)
proof −
  have tt > 0
    using assms(1,7) exec-pos-less-TT by simp
  then have tt − 1 < TT − exec t <#> jj
    using assms(6,7) by simp
  then have ∗: execute (tmL67 jj) (0, tps) (tt − 1) =
      (0, tpsL t xs (fmt n − tt + 1) 1 (λj. sim-move t ! j) (λj. sim-write t ! j))
    using assms execute-tmL67-1[where ?tt=tt − 1] by simp
  have ∗∗: fmt n − tt + 1 = exec t <#> jj
    using assms(1,6,7) exec-pos-less-TT Suc-diff-le less-eq-Suc-le by auto
  have execute (tmL67 jj) (0, tps) tt = exe (tmL67 jj) (execute (tmL67 jj) (0, tps) (tt − 1))
     using ‹tt > 0› exe-lt-length by (metis One-nat-def Suc-diff-Suc diff-zero execute.simps(2))
  also have ... = sem (cmdL7 jj) (execute (tmL67 jj) (0, tps) (tt − 1))
    using tmL67-def exe-lt-length ∗ by simp
  also have ... = sem (cmdL7 jj) (0, tpsL t xs (fmt n − tt + 1) 1 (λj. sim-move t ! j) (λj. sim-write t ! j))
    using ∗ by simp
   also have ... = (0, tpsL t (xs[jj:=(1, None)]) (fmt n − tt) 1 (λj. if j = jj then 3 else sim-move t ! j) (λj.
sim-write t ! j))
    using ∗∗ assms sem-cmdL7-2a[where ?i=fmt n − tt + 1] by simp
  finally show ?thesis
    using assms(8) by simp
```

qed

lemma *zip-cont-Some-Some*:
  assumes *jj* < *k*
    and *length xs* = *k*
    and *xs* ! *jj* = (*1*, *Some 0*)
    and *i* = *exec t* <#> *jj*
    and *i* = *0*
    and *sim-move t* ! *jj* = *0*
  shows *zip-cont t xs* = *zip-cont t* (*xs*[*jj*:=(*1*, *Some 1*)])
    (**is** *?lhs* = *?rhs*)
proof
  fix *p*
  consider *p* < *TT* | *p* ≥ *TT*
    by *linarith*
  then show *?lhs p* = *?rhs p*
  proof (*cases*)
    case *1*
    then have *?lhs p* = *zip-cont t xs p*
      by *simp*
    moreover have *zip-cont t xs p* = *zip-cont t* (*xs*[*jj*:=(*1*, *Some 1*)]) *p*
    proof (*rule zip-cont-eqI*)
      show *p* < *TT*
        using *1* **by** *simp*
      show ⋀*j*. *j* < *k* ⟹
          ((*exec* (*t* + *fst* (*xs* ! *j*))) <:> *j*) *p* =
          ((*exec* (*t* + *fst* (*xs*[*jj* := (*1*, *Some 1*)] ! *j*))) <:> *j*) *p*
        by (*metis assms*(*2*,*3*) *fst-conv nth-list-update-eq nth-list-update-neq*)
      show ⋀*j*. *j* < *k* ⟹
          (*case snd* (*xs* ! *j*) *of None* ⇒ *0* | *Some d* ⇒ *if p* = *exec* (*t* + *d*) <#> *j then 1 else 0*) =
          (*case snd* (*xs*[*jj* := (*1*, *Some 1*)] ! *j*) *of None* ⇒ *0* | *Some d* ⇒ *if p* = *exec* (*t* + *d*) <#> *j then 1 else 0*)
        using *assms 1 sim-move*
        by (*metis* (*no-types*, *lifting*) *add.commute add.right-neutral diff-add-zero nth-list-update-eq*
          nth-list-update-neq option.simps(*5*) *plus-1-eq-Suc prod.sel*(*2*))
    qed
    ultimately show *?thesis*
      by *simp*
  next
    case *2*
    then show *?thesis*
      using *zip-cont-eq-0* **by** *simp*
  qed
qed

lemma *sem-cmdL7-2b*:
  assumes *jj* < *k*
    and *tps* = *tpsL t xs i 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
    and *length xs* = *k*
    and *xs* ! *jj* = (*1*, *Some 0*)
    and *i* = *exec t* <#> *jj*
    and *i* = *0*
    and *sim-move t* ! *jj* = *0*
    and *tps'* = *tpsL t* (*xs*[*jj*:=(*1*, *Some 1*)]) *i 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
  shows *sem* (*cmdL7 jj*) (*0*, *tps*) = (*1*, *tps'*)
proof (*rule semI*[*of 2* ∗ *k* + *3*])
  show *proper-command* (*2* ∗ *k* + *3*) (*cmdL7 jj*)
    using *turing-command-cmdL7*[*OF assms*(*1*)] *turing-commandD*(*1*) **by** *simp*
  show *len*: *length tps* = *2* ∗ *k* + *3*
    using *assms*(*2*) **by** *simp*
  show *length tps'* = *2* ∗ *k* + *3*
    using *assms*(*8*) **by** *simp*

  define *rs* **where** *rs* = *read tps*

then have *is-beginning*: *is-beginning* (*rs* ! *1*)
  using *read-tpsL-1-nth-2k1 assms*(*2*,*6*) *enc-nth-def read-tpsL-1-bounds rs-def* **by** *simp*
then show *fst* (*cmdL7 jj* (*read tps*)) = *1*
  using *assms*(*6*) *cmdL7-def rs-def* **by** *simp*

have *rs* ! (*3* + *jj*) = □
  by (*simp add*: *rs-def assms add.commute read-tpsL-3'*)
then have *condition7c rs jj*
  using *is-beginning* **by** *simp*
then have ∗: *snd* (*cmdL7 jj rs*) =
 [(*rs* ! *0*, *Stay*),
  (*rs* ! *1*, *Left*),
  (*rs* ! *2*, *Stay*)] @
  (*map* (λ*j*. (*rs* ! (*j* + *3*), *Stay*)) [*0*..<*k*]) @
  (*map* (λ*j*. (*rs* ! (*3* + *k* + *j*), *Stay*)) [*0*..<*k*])
  unfolding *cmdL7-def* **by** *auto*

show *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *tps'* ! *j* **if** *j* < *2* ∗ *k* + *3* **for** *j*
proof −
  consider *j* = *0* | *j* = *1* | *j* = *2* | *3* ≤ *j* ∧ *j* < *k* + *3* | *k* + *3* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
    using ‹*j* < *2* ∗ *k* + *3*› **by** *linarith*
  then show *?thesis*
  proof (*cases*)
    case *1*
    then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL7 jj* (*read tps*) [!] *0*) (*tps* ! *0*)
      by *simp*
    also have ... = *act* (*rs* ! *0*, *Stay*) (*tps* ! *0*)
      using ∗ *rs-def* **by** *simp*
    also have ... = *tps* ! *0*
      using *act-Stay len rs-def* **by** *simp*
    also have ... = *tps'* ! *0*
      using *assms*(*2*,*8*) *tpsL-0* **by** *simp*
    also have ... = *tps'* ! *j*
      using *1* **by** *simp*
    finally show *?thesis* **.**
  next
    case *2*
    then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL7 jj* (*read tps*) [!] *1*) (*tps* ! *1*)
      by *simp*
    also have ... = *act* (*rs* ! *1*, *Left*) (*tps* ! *1*)
      using ∗ *rs-def* **by** *simp*
    also have ... = *tps'* ! *1*
      using *zip-cont-Some-Some assms rs-def tpsL-1 2 act-Left fst-conv len that tpsL-pos-1* **by** (*metis zero-diff*)
    finally show *?thesis*
      using *2* **by** *simp*
  next
    case *3*
    then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL7 jj* (*read tps*) [!] *2*) (*tps* ! *2*)
      by *simp*
    also have ... = *act* (*rs* ! *2*, *Stay*) (*tps* ! *2*)
      using ∗ *rs-def* **by** *simp*
    also have ... = *tps* ! *2*
      using *act-Stay len rs-def* **by** *simp*
    also have ... = *tps'* ! *2*
      using *assms*(*2*,*8*) *tpsL-2* **by** *simp*
    also have ... = *tps'* ! *j*
      using *3* **by** *simp*
    finally show *?thesis* **.**
  next
    case *4*
    then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*rs* ! *j*, *Stay*) (*tps* ! *j*)
      using ∗ *rs-def threeplus2k-2*[**where** *?a*=(*rs* ! *0*, *Stay*)] *4 diff-add-inverse* **by** *auto*
    also have ... = *tps'* ! *j*

```
      using 4 assms(2,8) act-Stay len rs-def that tpsL-mvs′ by (metis add.commute)
    finally show ?thesis .
  next
    case 5
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
      using ∗ rs-def threeplus2k-3[where ?a=(rs ! 0, Stay)] by simp
    also have ... = tps ! j
      using len act-Stay rs-def that by simp
    also have ... = tps′ ! j
      using assms(2,8) tpsL-symbs′ 5 by simp
    finally show ?thesis .
  qed
 qed
qed

lemma execute-tmL67-2b:
 assumes jj < k
  and tps = tpsL t xs (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  and length xs = k
  and xs ! jj = (1, Some 0)
  and sim-move t ! jj = 0
  and exec t <#> jj = 0
  and tps′ = tpsL t (xs[jj:=(1, Some 1)]) 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
 shows execute (tmL67 jj) (0, tps) TT = (1, tps′)
 using execute-tmL67-1 assms exe-lt-length tmL67-def sem-cmdL7-2b by simp

lemma tmL67-left-0:
 assumes jj < k
  and tps = tpsL t xs (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  and length xs = k
  and xs ! jj = (1, Some 0)
  and sim-move t ! jj = 0
  and exec t <#> jj = 0
  and tps′ = tpsL t (xs[jj:=(1, Some 1)]) 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
 shows traces (tmL67 jj) tps esL67 tps′
proof
 show execute (tmL67 jj) (0, tps) (length esL67) = (length (tmL67 jj), tps′)
  using esL67-def tmL67-def execute-tmL67-2b assms by simp
 show ⋀i. i < length esL67 ⟹
     fst (execute (tmL67 jj) (0, tps) i) < length (tmL67 jj)
  using esL67-def tmL67-def execute-tmL67-1 assms by simp
 show execute (tmL67 jj) (0, tps) (Suc i) <#> 0 = fst (esL67 ! i) ∧
     execute (tmL67 jj) (0, tps) (Suc i) <#> 1 = snd (esL67 ! i)
  if i < length esL67 for i
 proof (cases i = fmt n)
  case True
  then have Suc i = TT
    by simp
  moreover have esL67 ! i = (n + 1, 0)
    using True esL67-def by (simp add: nth-append)
  ultimately show ?thesis
    using assms that tpsL-pos-0 tpsL-pos-1 by (metis execute-tmL67-2b fst-conv snd-conv)
 next
  case False
  then have Suc i < TT
    using that esL67-def by simp
  moreover from this have esL67 ! i = (n + 1, fmt n − 1 − i)
    by simp
  ultimately show ?thesis
    using tpsL-pos-0 tpsL-pos-1 assms(1−6) execute-tmL67-1
    by (metis (no-types, lifting) diff-diff-left fst-conv minus-nat.diff-0 plus-1-eq-Suc snd-conv)
 qed
qed
```

```
lemma sem-cmdL7-3:
  assumes jj < k
    and tps = tpsL t xs i 1 (λj. if j = jj then 3 else sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (1, None)
    and Suc i = exec t <#> jj
    and sim-move t ! jj = 0
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) (i − 1) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows sem (cmdL7 jj) (0, tps) = (if i = 0 then 1 else 0, tps')
proof (rule semI[of 2 ∗ k + 3])
  show proper-command (2 ∗ k + 3) (cmdL7 jj)
    using turing-command-cmdL7[OF assms(1)] turing-commandD(1) by simp
  show len: length tps = 2 ∗ k + 3
    using assms(2) by simp
  show length tps' = 2 ∗ k + 3
    using assms(7) by simp

  define rs where rs = read tps
  show fst (cmdL7 jj (read tps)) = (if i = 0 then 1 else 0)
  proof (cases i = 0)
    case True
    then have is-beginning (rs ! 1)
      using read-tpsL-1-nth-2k1 assms(2) enc-nth-def read-tpsL-1-bounds rs-def by simp
    then show ?thesis
      using True cmdL7-def rs-def by simp
  next
    case False
    then have ¬ is-beginning (rs ! 1)
      using read-tpsL-1-nth-2k1 assms enc-nth-def exec-pos-less-TT
      by (metis (no-types, lifting) Suc-le-lessD less-imp-le-nat less-numeral-extra(1) neq0-conv rs-def)
    then show ?thesis
      using False cmdL7-def rs-def by simp
  qed

  have i < TT
    using assms exec-pos-less-TT by (metis Suc-less-eq less-SucI)

  have rs ! (3 + jj) = 3
    by (simp add: rs-def assms(1,2) add.commute read-tpsL-3')
  moreover have is-code (rs ! 1)
    using assms rs-def read-tpsL-1-nth-less-2k ‹i < TT› read-tpsL-1-bounds by simp
  ultimately have condition7b rs jj
    using ‹i < TT› assms(2) read-tpsL-1-bounds rs-def by simp
  then have ∗: snd (cmdL7 jj rs) =
    [(rs ! 0, Stay),
     (enc-upd (rs ! 1) (k + jj) 1, Left),
     (rs ! 2, Stay)] @
     (map (λj. (if j = jj then 0 else rs ! (j + 3), Stay)) [0..<k]) @
     (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
    unfolding cmdL7-def by simp

  show act (cmdL7 jj (read tps) [!] j) (tps ! j) = tps' ! j if j < 2 ∗ k + 3 for j
  proof −
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 ∗ k + 3
      using ‹j < 2 ∗ k + 3› by linarith
    then show ?thesis
    proof (cases)
      case 1
      then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 0) (tps ! 0)
        by simp
      also have ... = act (rs ! 0, Stay) (tps ! 0)
        using ∗ rs-def by simp
```

also have ... = *tps* ! *0*
  using *act-Stay len rs-def* **by** *simp*
also have ... = *tps'* ! *0*
  using *assms*(*2*,*7*) *tpsL-0* **by** *simp*
also have ... = *tps'* ! *j*
  using *1* **by** *simp*
finally show *?thesis* **.**
next
case *2*
then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL7 jj* (*read tps*) [!] *1*) (*tps* ! *1*)
  by *simp*
also have ... = *act* (*enc-upd* (*rs* ! *1*) (*k* + *jj*) *1*, *Left*) (*tps* ! *1*)
  using ∗ *rs-def* **by** *simp*
also have ... = *tps* ! *1* |:=| (*enc-upd* (*rs* ! *1*) (*k* + *jj*) *1*) |−| *1*
  using *act-Left' 2 len* **by** *simp*
also have ... = *tps'* ! *1*
  using *assms zip-cont-enc-upd-Some-Left rs-def read-tpsL-1 tpsL-1 zip-cont-def* **by** *simp*
finally show *?thesis*
  using *2* **by** *simp*
next
case *3*
then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL7 jj* (*read tps*) [!] *2*) (*tps* ! *2*)
  by *simp*
also have ... = *act* (*rs* ! *2*, *Stay*) (*tps* ! *2*)
  using ∗ *rs-def* **by** *simp*
also have ... = *tps* ! *2*
  using *act-Stay len rs-def* **by** *simp*
also have ... = *tps'* ! *2*
  using *assms*(*2*,*7*) *tpsL-2* **by** *simp*
also have ... = *tps'* ! *j*
  using *3* **by** *simp*
finally show *?thesis* **.**
next
case *4*
show *?thesis*
proof (*cases j* = *3* + *jj*)
  case *True*
  then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*0*, *Stay*) (*tps* ! *j*)
    using ∗ *rs-def threeplus2k-2*[**where** *?a*=(*rs* ! *0*, *Stay*)] *4 diff-add-inverse*
    by (*smt* (*verit*, *ccfv-threshold*))
  also have ... = *tps'* ! *j*
    using *4 assms*(*1*,*2*,*6*,*7*) *4 True act-onesie tpsL-mvs* **by** *simp*
  finally show *?thesis* **.**
next
  case *False*
  then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*rs* ! *j*, *Stay*) (*tps* ! *j*)
    using ∗ *rs-def threeplus2k-2*[**where** *?a*=(*rs* ! *0*, *Stay*)] *4 diff-add-inverse* **by** *auto*
  also have ... = *tps'* ! *j*
    using *4 assms*(*2*,*7*) *False act-Stay len rs-def that tpsL-mvs'*
    by (*smt* (*verit*) *add.commute le-add-diff-inverse2*)
  finally show *?thesis* **.**
qed
next
case *5*
then have *act* (*cmdL7 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*rs* ! *j*, *Stay*) (*tps* ! *j*)
  using ∗ *rs-def threeplus2k-3*[**where** *?a*=(*rs* ! *0*, *Stay*)] **by** *simp*
also have ... = *tps* ! *j*
  using *len act-Stay rs-def that* **by** *simp*
also have ... = *tps'* ! *j*
  using *assms*(*2*,*7*) *tpsL-symbs' 5* **by** *simp*
finally show *?thesis* **.**
qed
qed

qed

lemma *execute-tmL67-3*:
  assumes $jj < k$
    and $tps = tpsL\ t\ xs\ (fmt\ n)\ 1\ (\lambda j.\ sim\text{-}move\ t\ !\ j)\ (\lambda j.\ sim\text{-}write\ t\ !\ j)$
    and $length\ xs = k$
    and $xs\ !\ jj = (1,\ Some\ 0)$
    and $sim\text{-}move\ t\ !\ jj = 0$
    and $exec\ t <\#> jj > 0$
    and $tt = TT - exec\ t <\#> jj$
    and $tps' = tpsL\ t\ (xs[jj:=(1,\ Some\ 1)])\ (fmt\ n - Suc\ tt)\ 1\ (\lambda j.\ sim\text{-}move\ t\ !\ j)\ (\lambda j.\ sim\text{-}write\ t\ !\ j)$
  shows $execute\ (tmL67\ jj)\ (0,\ tps)\ (Suc\ tt) = (if\ fmt\ n - tt = 0\ then\ 1\ else\ 0,\ tps')$
proof −
  let $?i = fmt\ n - tt$
  let $?xs = xs[jj:=(1,\ None)]$
  let $?tps = tpsL\ t\ ?xs\ ?i\ 1\ (\lambda j.\ if\ j = jj\ then\ 3\ else\ sim\text{-}move\ t\ !\ j)\ (\lambda j.\ sim\text{-}write\ t\ !\ j)$
  have *1*: $Suc\ ?i = exec\ t <\#> jj$
    using *assms exec-pos-less-TT*
    by (*smt* (*verit*) *Suc-diff-le diff-diff-cancel diff-is-0-eq nat-less-le neq0-conv not-less-eq zero-less-diff*)
  have *2*: $?xs\ !\ jj = (1,\ None)$
    by (*simp add*: *assms*(*1*) *assms*(*3*))
  have *3*: $length\ ?xs = k$
    by (*simp add*: *assms*(*3*))
  have $execute\ (tmL67\ jj)\ (0,\ tps)\ (Suc\ tt) = exe\ (tmL67\ jj)\ (execute\ (tmL67\ jj)\ (0,\ tps)\ tt)$
    by *simp*
  also have ... $= exe\ (tmL67\ jj)\ (0,\ ?tps)$
    using *assms execute-tmL67-2a* **by** *simp*
  also have ... $= sem\ (cmdL7\ jj)\ (0,\ ?tps)$
    using *tmL67-def exe-lt-length* **by** *simp*
  also have ... $= (if\ fmt\ n - tt = 0\ then\ 1\ else\ 0,\ tps')$
    using *sem-cmdL7-3*[*OF assms*(*1*) *- 3 2 1 assms*(*5*)] *assms*(*8*) **by** *simp*
  finally show *?thesis*
    by *simp*
qed

lemma *sem-cmdL7-4*:
  assumes $jj < k$
    and $tps = tpsL\ t\ xs\ i\ 1\ (\lambda j.\ sim\text{-}move\ t\ !\ j)\ (\lambda j.\ sim\text{-}write\ t\ !\ j)$
    and $length\ xs = k$
    and $xs\ !\ jj = (1,\ Some\ 1)$
    and $Suc\ i < exec\ t <\#> jj$
    and $sim\text{-}move\ t\ !\ jj = 0$
    and $tps' = tpsL\ t\ xs\ (i - 1)\ 1\ (\lambda j.\ sim\text{-}move\ t\ !\ j)\ (\lambda j.\ sim\text{-}write\ t\ !\ j)$
  shows $sem\ (cmdL7\ jj)\ (0,\ tps) = (if\ i = 0\ then\ 1\ else\ 0,\ tps')$
proof (*rule semI*[*of* $2 * k + 3$])
  show $proper\text{-}command\ (2 * k + 3)\ (cmdL7\ jj)$
    using *turing-command-cmdL7*[*OF assms*(*1*)] *turing-commandD*(*1*) **by** *simp*
  show *len*: $length\ tps = 2 * k + 3$
    using *assms*(*2*) **by** *simp*
  show $length\ tps' = 2 * k + 3$
    using *assms*(*7*) **by** *simp*

  define *rs* **where** $rs = read\ tps$
  show $fst\ (cmdL7\ jj\ (read\ tps)) = (if\ i = 0\ then\ 1\ else\ 0)$
  proof (*cases* $i = 0$)
    case *True*
    then have $is\text{-}beginning\ (rs\ !\ 1)$
      using *read-tpsL-1-nth-2k1 assms*(*2*) *enc-nth-def read-tpsL-1-bounds rs-def* **by** *simp*
    then show *?thesis*
      using *True cmdL7-def rs-def* **by** *simp*
  next
    case *False*
    then have $\neg\ is\text{-}beginning\ (rs\ !\ 1)$

```
    using read-tpsL-1-nth-2k1 assms enc-nth-def exec-pos-less-TT read-tpsL-1 rs-def
    by (metis (no-types, lifting) less-2-cases-iff nat-1-add-1 not-less-eq plus-1-eq-Suc)
  then show ?thesis
    using False cmdL7-def rs-def by simp
qed

have i < exec t <#> jj
  using assms(5) by simp
then have i < TT
  using assms(1) exec-pos-less-TT by (meson Suc-lessD less-trans-Suc)

have rs ! (3 + jj) = □
  using rs-def read-tpsL-3 assms by simp
moreover have enc-nth (rs ! 1) (k + jj) = 0
  using assms rs-def read-tpsL-1-nth-less-2k[OF ‹i < TT›, of k + jj] sim-move by simp
ultimately have condition7c rs jj
  by simp
then have *: snd (cmdL7 jj rs) =
  [(rs ! 0, Stay),
   (rs ! 1, Left),
   (rs ! 2, Stay)] @
   (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
   (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
  unfolding cmdL7-def by auto

show act (cmdL7 jj (read tps) [!] j) (tps ! j) = tps' ! j if j < 2 * k + 3 for j
proof -
  consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
    using ‹j < 2 * k + 3› by linarith
  then show ?thesis
  proof (cases)
    case 1
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 0) (tps ! 0)
      by simp
    also have ... = act (rs ! 0, Stay) (tps ! 0)
      using * rs-def by simp
    also have ... = tps ! 0
      using act-Stay len rs-def by simp
    also have ... = tps' ! 0
      using assms(2,7) tpsL-0 by simp
    also have ... = tps' ! j
      using 1 by simp
    finally show ?thesis .
  next
    case 2
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 1) (tps ! 1)
      by simp
    also have ... = act (rs ! 1, Left) (tps ! 1)
      using * rs-def by simp
    also have ... = tps' ! 1
      using assms rs-def tpsL-1 2 act-Left fst-conv len that tpsL-pos-1 by metis
    finally show ?thesis
      using 2 by simp
  next
    case 3
    then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (cmdL7 jj (read tps) [!] 2) (tps ! 2)
      by simp
    also have ... = act (rs ! 2, Stay) (tps ! 2)
      using * rs-def by simp
    also have ... = tps ! 2
      using act-Stay len rs-def by simp
    also have ... = tps' ! 2
      using assms(2,7) tpsL-2 by simp
```

```
      also have ... = tps' ! j
        using 3 by simp
      finally show ?thesis .
    next
      case 4
      then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
        using * rs-def threeplus2k-2[where ?a=(rs ! 0, Stay)] 4 diff-add-inverse by auto
      also have ... = tps' ! j
        using 4 assms(2,7) act-Stay len rs-def that tpsL-mvs' by (smt (verit) add.commute le-add-diff-inverse2)
      finally show ?thesis .
    next
      case 5
      then have act (cmdL7 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
        using * rs-def threeplus2k-3[where ?a=(rs ! 0, Stay)] by simp
      also have ... = tps ! j
        using len act-Stay rs-def that by simp
      also have ... = tps' ! j
        using assms(2,7) tpsL-symbs' 5 by simp
      finally show ?thesis .
    qed
  qed
qed

lemma execute-tmL67-4:
  assumes jj < k
    and tps = tpsL t xs (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (1, Some 0)
    and sim-move t ! jj = 0
    and exec t <#> jj > 0
    and tt ≥ Suc (Suc (TT − exec t <#> jj))
    and tt ≤ TT
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) (fmt n − tt) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows execute (tmL67 jj) (0, tps) tt = (if TT − tt = 0 then 1 else 0, tps')
  using assms(7,8,9)
proof (induction tt arbitrary: tps' rule: nat-induct-at-least)
  case base
  let ?tt = TT − exec t <#> jj
  let ?xs = xs[jj:=(1, Some 1)]
  let ?tps = tpsL t ?xs (fmt n − Suc ?tt) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  have execute (tmL67 jj) (0, tps) (Suc (Suc ?tt)) = exe (tmL67 jj) (execute (tmL67 jj) (0, tps) (Suc ?tt))
    by simp
  also have ... = exe (tmL67 jj) (if fmt n − ?tt = 0 then 1 else 0, ?tps)
    using execute-tmL67-3 assms by simp
  also have ... = sem (cmdL7 jj) (0, ?tps)
    using tmL67-def base exe-lt-length by simp
  finally show ?case
    using sem-cmdL7-4 assms base.prems(2) by simp
next
  case (Suc tt)
  let ?tt = TT − exec t <#> jj
  let ?xs = xs[jj:=(1, Some 1)]
  let ?tps = tpsL t ?xs (fmt n − tt) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  have execute (tmL67 jj) (0, tps) (Suc tt) = exe (tmL67 jj) (execute (tmL67 jj) (0, tps) tt)
    by simp
  also have ... = exe (tmL67 jj) (if Suc (fmt n) − tt = 0 then 1 else 0, ?tps)
    using Suc by simp
  also have ... = sem (cmdL7 jj) (0, ?tps)
    using Suc.prems(1) exe-lt-length tmL67-def by auto
  finally show ?case
    using assms sem-cmdL7-4 Suc by simp
qed
```

```
lemma tmL67-left-gt-0:
  assumes jj < k
    and tps = tpsL t xs (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (1, Some 0)
    and sim-move t ! jj = 0
    and exec t <#> jj > 0
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL67 jj) tps esL67 tps'
proof
  show 1: execute (tmL67 jj) (0, tps) (length esL67) = (length (tmL67 jj), tps')
  proof (cases exec t <#> jj = 1)
    case True
    then show ?thesis
      using assms(7) execute-tmL67-3[OF assms(1−6)] esL67-def tmL67-def by simp
  next
    case False
    then have TT ≥ Suc (Suc (TT − exec t <#> jj))
      using assms(1,6,7) exec-pos-less-TT
      by (metis Suc-leI add-diff-cancel-right' diff-diff-cancel diff-less nat-less-le plus-1-eq-Suc zero-less-Suc)
    then show ?thesis
      using assms(7) execute-tmL67-4[OF assms(1−6), where ?tt=TT] esL67-def tmL67-def by simp
  qed
  show fst (execute (tmL67 jj) (0, tps) tt) < length (tmL67 jj) if tt < length esL67 for tt
  proof −
    have tt < TT
      using that esL67-def by simp
    then consider
        tt < TT − exec t <#> jj
      | tt = TT − exec t <#> jj
      | tt = Suc (TT − exec t <#> jj)
      | tt ≥ Suc (Suc (TT − exec t <#> jj))
      by linarith
    then show ?thesis
    proof (cases)
      case 1
      then show ?thesis
        using assms execute-tmL67-1 tmL67-def by simp
    next
      case 2
      then show ?thesis
        using assms execute-tmL67-2a tmL67-def by simp
    next
      case 3
      then show ?thesis
        using assms execute-tmL67-3 tmL67-def ‹tt < TT› by simp
    next
      case 4
      then show ?thesis
        using assms execute-tmL67-4 tmL67-def ‹tt < TT› by simp
    qed
  qed
  show execute (tmL67 jj) (0, tps) (Suc tt) <#> 0 = fst (esL67 ! tt) ∧
      execute (tmL67 jj) (0, tps) (Suc tt) <#> 1 = snd (esL67 ! tt)
    if tt < length esL67 for tt
  proof −
    have *: Suc tt ≤ TT
      using that esL67-def by simp
    then consider
        Suc tt < TT − exec t <#> jj
      | Suc tt = TT − exec t <#> jj
      | Suc tt = Suc (TT − exec t <#> jj)
      | Suc tt ≥ Suc (Suc (TT − exec t <#> jj))
```

```
    using esL67-def ‹tt < length esL67› by linarith
  then show ?thesis
  proof (cases)
    case 1
    then show ?thesis
      using execute-tmL67-1[OF assms(1−5), where ?tt=Suc tt] tmL67-def tpsL-pos-0 tpsL-pos-1 ∗
      by simp
  next
    case 2
    then show ?thesis
      using assms execute-tmL67-2a[OF assms(1−6), where ?tt=Suc tt] tmL67-def tpsL-pos-0 tpsL-pos-1 ∗
      by simp
  next
    case 3
    then show ?thesis
      using assms(6) execute-tmL67-3[OF assms(1−6), where ?tt=tt] tmL67-def tpsL-pos-0 tpsL-pos-1 ∗
      by (smt (verit, ccfv-threshold) add.commute diff-Suc-1 diff-diff-left diff-is-0-eq
        esL67-at-fmtn esL67-at-lt-fmtn nat-less-le plus-1-eq-Suc prod.collapse prod.inject)
  next
    case 4
    then show ?thesis
      using assms(7) execute-tmL67-4[OF assms(1−6) - ∗] ∗ tmL67-def tpsL-pos-0 tpsL-pos-1 1 esL67-at-fmtn
esL67-at-lt-fmtn esL67-def
      by (smt (verit, best) One-nat-def Suc-diff-Suc add.commute diff-Suc-1 fst-conv le-neq-implies-less
        length-append length-map length-rev length-upt list.size(3) list.size(4) minus-nat.diff-0 not-less-eq
        plus-1-eq-Suc snd-conv)
  qed
 qed
qed

lemma tmL67-left:
 assumes jj < k
  and tps = tpsL t xs (fmt n) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  and length xs = k
  and xs ! jj = (1, Some 0)
  and sim-move t ! jj = 0
  and tps' = tpsL t (xs[jj:=(1, Some 1)]) 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
 shows traces (tmL67 jj) tps esL67 tps'
 using assms tmL67-left-0 tmL67-left-gt-0 by (cases exec t <#> jj = 0) simp-all

definition esL47 ≡ esL46 @ esL67

lemma len-esL47: length esL47 = 2 ∗ TT + 2
 using len-esL46 esL47-def esL67-def by simp

lemma tmL47-nonleft:
 assumes jj < k
  and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  and length xs = k
  and xs ! jj = (0, Some 0)
  and sim-move t ! jj ≠ 0
  and tps' = tpsL t (xs[jj:=(1, Some 0)]) 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
 shows traces (tmL47 jj) tps esL47 tps'
 unfolding tmL47-def esL47-def
 using assms
 by (intro traces-sequential[OF tmL46 tmL67-nonleft]) simp-all

lemma tmL47-left:
 assumes jj < k
  and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  and length xs = k
  and xs ! jj = (0, Some 0)
  and sim-move t ! jj = 0
```

```
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL47 jj) tps esL47 tps'
  unfolding tmL47-def esL47-def
  using assms
  by (intro traces-sequential[OF tmL46 tmL67-left[where ?xs=xs[jj:=(1, Some 0)]]]) simp-all

lemma sem-cmdL8-nonright:
  assumes jj < k
    and tps = tpsL t xs i 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and i < TT
    and sim-move t ! jj ≠ 2
    and tps' = tpsL t xs (Suc i) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows sem (cmdL8 jj) (0, tps) = (0, tps')
proof (rule semI[of 2 ∗ k + 3])
  show proper-command (2 ∗ k + 3) (cmdL8 jj)
    using turing-command-cmdL8[OF assms(1)] turing-commandD(1) by simp
  show len: length tps = 2 ∗ k + 3
    using assms(2) by simp
  show length tps' = 2 ∗ k + 3
    using assms(6) by simp
  define rs where rs = read tps
  then have rs ! 1 ≠ □
    using assms by (metis not-one-less-zero read-tpsL-1-bounds(1))
  then show fst (cmdL8 jj rs) = 0
    unfolding cmdL8-def by simp

  have rs ! (3 + jj) = sim-move t ! jj
    using rs-def assms(1,2) read-tpsL-3 by simp
  moreover have sim-move t ! jj < 3
    using sim-move-def assms(1) direction-to-symbol-less sim-move-nth sim-move-nth-else
    by (metis One-nat-def not-add-less2 not-less-eq numeral-3-eq-3 plus-1-eq-Suc)
  ultimately have condition8d rs jj
    using assms(5) ‹rs ! 1 ≠ □› by simp
  then have ∗: snd (cmdL8 jj rs) =
    [(rs ! 0, Stay),
     (rs ! 1, Right),
     (rs ! 2, Stay)] @
     (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
     (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
    unfolding cmdL8-def by auto

  show act (cmdL8 jj (read tps) [!] j) (tps ! j) = tps' ! j if j < 2 ∗ k + 3 for j
  proof −
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 ∗ k + 3
      using ‹j < 2 ∗ k + 3› by linarith
    then show ?thesis
    proof (cases)
      case 1
      then show ?thesis
        by (metis ∗ act-Stay append.simps(2) assms(2) assms(6) len nth-Cons-0 rs-def that tpsL-0)
    next
      case 2
      then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (cmdL8 jj (read tps) [!] 1) (tps ! 1)
        by simp
      also have ... = act (rs ! 1, Right) (tps ! 1)
        using ∗ rs-def by simp
      also have ... = tps' ! 1
        using act-Right len rs-def assms tpsL-1 that tpsL-pos-1
        by (metis 2 add.commute fst-conv plus-1-eq-Suc)
      also have ... = tps' ! j
        using 2 by simp
      finally show ?thesis .
```

```
    next
      case 3
      then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (cmdL8 jj (read tps) [!] 2) (tps ! 2)
        by simp
      also have ... = act (rs ! 2, Stay) (tps ! 2)
        using * rs-def by simp
      also have ... = tps ! 2
        using act-Stay len rs-def by simp
      also have ... = tps' ! 2
        using assms(2,6) tpsL-2 by simp
      also have ... = tps' ! j
        using 3 by simp
      finally show ?thesis .
    next
      case 4
      then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
        using * rs-def threeplus2k-2[where ?a=(rs ! 0, Stay)] by simp
      also have ... = tps ! j
        using len act-Stay rs-def that by simp
      also have ... = tps' ! j
        using assms(2,6) tpsL-mvs' 4 by simp
      finally show ?thesis .
    next
      case 5
      then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
        using * rs-def threeplus2k-3[where ?a=(rs ! 0, Stay)] by simp
      also have ... = tps ! j
        using len act-Stay rs-def that by simp
      also have ... = tps' ! j
        using assms(2,6) tpsL-symbs' 5 by simp
      finally show ?thesis .
    qed
  qed
qed

lemma sem-cmdL8-TT:
  assumes jj < k
    and tps = tpsL t xs i 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and i = TT
  shows sem (cmdL8 jj) (0, tps) = (1, tps)
proof (rule semI[of 2 * k + 3])
  show proper-command (2 * k + 3) (cmdL8 jj)
    using turing-command-cmdL8[OF assms(1)] turing-commandD(1) by simp
  show len: length tps = 2 * k + 3
    using assms(2) by simp
  show length tps = 2 * k + 3
    using assms(2) by simp
  define rs where rs = read tps
  then have rs ! 1 = □
    using assms read-tpsL-1 by simp
  then show fst (cmdL8 jj rs) = 1
    unfolding cmdL8-def by simp

  have rs ! (3 + jj) = sim-move t ! jj
    using rs-def assms(1,2) read-tpsL-3 by simp
  moreover have sim-move t ! jj < 3
    using sim-move-def assms(1) direction-to-symbol-less sim-move-nth sim-move-nth-else
    by (metis One-nat-def not-add-less2 not-less-eq numeral-3-eq-3 plus-1-eq-Suc)
  ultimately have condition8c rs jj
    using ‹rs ! 1 = □› by simp
  then have *: snd (cmdL8 jj rs) =
    [(rs ! 0, Stay),
```

```
    (rs ! 1, Stay),
    (rs ! 2, Stay)] @
    (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
    (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
   unfolding cmdL8-def by simp

  show act (cmdL8 jj (read tps) [!] j) (tps ! j) = tps ! j if j < 2 * k + 3 for j
  proof −
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
      using ‹j < 2 * k + 3› by linarith
    then show ?thesis
      using * act-Stay len rs-def
        threeplus2k-2[where ?a=(rs ! 0, Stay)] threeplus2k-3[where ?a=(rs ! 0, Stay)]
      by (cases) simp-all
  qed
qed

lemma execute-tmL78-nonright-le-TT:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and sim-move t ! jj ≠ 2
    and tt ≤ TT
    and tps' = tpsL t xs tt 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows execute (tmL78 jj) (0, tps) tt = (0, tps')
  using assms(5,6)
proof (induction tt arbitrary: tps')
  case 0
  then show ?case
    using assms(1−4) by simp
next
  case (Suc tt)
  then have tt < TT
    by simp
  have execute (tmL78 jj) (0, tps) (Suc tt) = exe (tmL78 jj) (execute (tmL78 jj) (0, tps) tt)
    by simp
  also have ... = exe (tmL78 jj) (0, tpsL t xs tt 1 (λj. sim-move t ! j) (λj. sim-write t ! j))
    using Suc by simp
  also have ... = sem (cmdL8 jj) (0, tpsL t xs tt 1 (λj. sim-move t ! j) (λj. sim-write t ! j))
    using tmL78-def exe-lt-length by simp
  finally show ?case
    using sem-cmdL8-nonright[OF assms(1) - assms(3) ‹tt < TT› assms(4)] Suc by simp
qed

lemma execute-tmL78-nonright-eq-Suc-TT:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and sim-move t ! jj ≠ 2
    and tps' = tpsL t xs TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows execute (tmL78 jj) (0, tps) (Suc TT) = (1, tps')
  using assms sem-cmdL8-TT execute-tmL78-nonright-le-TT[where ?tt=TT] exe-lt-length tmL78-def
  by simp

definition esL78 ≡ map (λi. (n + 1, Suc i)) ([0..<TT]) @ [(n + 1, TT)]

lemma tmL78-nonright:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and sim-move t ! jj ≠ 2
    and tps' = tpsL t xs TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL78 jj) tps esL78 tps'
```

proof
  have *len*: *length esL78* = *Suc TT*
    using *esL78-def* **by** *simp*
  then show *1*: *execute* (*tmL78 jj*) (*0*, *tps*) (*length esL78*) = (*length* (*tmL78 jj*), *tps'*)
    using *assms tmL78-def execute-tmL78-nonright-eq-Suc-TT* **by** *simp*
  show $\bigwedge$*i*. *i* < *length esL78* $\Longrightarrow$
      fst (*execute* (*tmL78 jj*) (*0*, *tps*) *i*) < *length* (*tmL78 jj*)
    using *len assms execute-tmL78-nonright-le-TT tmL78-def* **by** *simp*
  show (*execute* (*tmL78 jj*) (*0*, *tps*) (*Suc i*)) <#> *0* = *fst* (*esL78* ! *i*) ∧
      (*execute* (*tmL78 jj*) (*0*, *tps*) (*Suc i*)) <#> *1* = *snd* (*esL78* ! *i*)
    if *i* < *length esL78* **for** *i*
  proof (*cases i* = *TT*)
    case *True*
    then have *esL78* ! *i* = (*n* + *1*, *TT*)
      using *esL78-def* **by** (*simp add*: *nth-append*)
    then show *?thesis*
      using *assms that tpsL-pos-0 tpsL-pos-1 len 1 True* **by** *simp*
  next
    case *False*
    then have *i* < *TT*
      using *that len* **by** *simp*
    moreover from *this* **have** *esL78* ! *i* = (*n* + *1*, *Suc i*)
      using *esL78-def nth-map-upt-TT* **by** *auto*
    ultimately show *?thesis*
      using *tpsL-pos-0 tpsL-pos-1 assms*(*1*−*4*) *execute-tmL78-nonright-le-TT*
      by (*metis Suc-leI fst-conv snd-conv*)
  qed
qed

lemma *sem-cmdL8-1*:
  assumes *jj* < *k*
    and *tps* = *tpsL t xs i 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
    and *length xs* = *k*
    and *xs* ! *jj* = (*1*, *Some 0*)
    and *i* < *exec t* <#> *jj*
    and *sim-move t* ! *jj* = *2*
    and *tps'* = *tpsL t xs* (*Suc i*) *1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
  shows *sem* (*cmdL8 jj*) (*0*, *tps*) = (*0*, *tps'*)
proof (*rule semI*[*of 2* ∗ *k* + *3*])
  show *proper-command* (*2* ∗ *k* + *3*) (*cmdL8 jj*)
    using *turing-command-cmdL8*[*OF assms*(*1*)] *turing-commandD*(*1*) **by** *simp*
  show *len*: *length tps* = *2* ∗ *k* + *3*
    using *assms*(*2*) **by** *simp*
  show *length tps'* = *2* ∗ *k* + *3*
    using *assms*(*7*) **by** *simp*

  have *i* < *TT*
    using *assms exec-pos-less-TT* **by** (*meson Suc-less-eq less-SucI less-trans-Suc*)

  define *rs* **where** *rs* = *read tps*
  then have *rs* ! *1* ≠ □
    using *assms* ‹*i* < *TT*› **by** (*metis not-one-less-zero read-tpsL-1-bounds*(*1*))
  then show *fst* (*cmdL8 jj rs*) = *0*
    unfolding *cmdL8-def* **by** *simp*

  have *rs* ! (*3* + *jj*) = *sim-move t* ! *jj*
    using *rs-def assms*(*1*,*2*) *read-tpsL-3* **by** *simp*
  moreover have *sim-move t* ! *jj* = *2*
    using *sim-move-def assms*(*1*,*6*) *direction-to-symbol-less sim-move-nth sim-move-nth-else*
    by *simp*
  moreover have *enc-nth* (*rs* ! *1*) (*k* + *jj*) = *0*
    using *assms rs-def read-tpsL-1-nth-less-2k*[*OF* ‹*i* < *TT*›, *of k* + *jj*] **by** *simp*
  ultimately have *condition8d rs jj*

```
  using assms ‹rs ! 1 ≠ □› by simp
then have *: snd (cmdL8 jj rs) =
  [(rs ! 0, Stay),
   (rs ! 1, Right),
   (rs ! 2, Stay)] @
   (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
   (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
  unfolding cmdL8-def by auto

show act (cmdL8 jj (read tps) [!] j) (tps ! j) = tps' ! j if j < 2 * k + 3 for j
proof -
  consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
    using ‹j < 2 * k + 3› by linarith
  then show ?thesis
  proof (cases)
    case 1
    then show ?thesis
      using * act-Stay append.simps(2) assms len nth-Cons-0 rs-def that tpsL-0 by metis
  next
    case 2
    then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (cmdL8 jj (read tps) [!] 1) (tps ! 1)
      by simp
    also have ... = act (rs ! 1, Right) (tps ! 1)
      using * rs-def by simp
    also have ... = tps' ! 1
      using act-Right len rs-def assms tpsL-1 that tpsL-pos-1
      by (metis 2 add.commute fst-conv plus-1-eq-Suc)
    also have ... = tps' ! j
      using 2 by simp
    finally show ?thesis .
  next
    case 3
    then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (cmdL8 jj (read tps) [!] 2) (tps ! 2)
      by simp
    also have ... = act (rs ! 2, Stay) (tps ! 2)
      using * rs-def by simp
    also have ... = tps ! 2
      using act-Stay len rs-def by simp
    also have ... = tps' ! 2
      using assms(2,7) tpsL-2 by simp
    also have ... = tps' ! j
      using 3 by simp
    finally show ?thesis .
  next
    case 4
    then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
      using * rs-def threeplus2k-2[where ?a=(rs ! 0, Stay)] by simp
    also have ... = tps ! j
      using len act-Stay rs-def that by simp
    also have ... = tps' ! j
      using assms(2,7) tpsL-mvs' 4 by simp
    finally show ?thesis .
  next
    case 5
    then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
      using * rs-def threeplus2k-3[where ?a=(rs ! 0, Stay)] by simp
    also have ... = tps ! j
      using len act-Stay rs-def that by simp
    also have ... = tps' ! j
      using assms(2,7) tpsL-symbs' 5 by simp
    finally show ?thesis .
  qed
qed
```

qed

lemma *execute-tmL78-1*:
assumes *jj* < *k*
and *tps* = *tpsL t xs 0 1* (*λj. sim-move t* ! *j*) (*λj. sim-write t* ! *j*)
and *length xs* = *k*
and *xs* ! *jj* = (*1*, *Some 0*)
and *sim-move t* ! *jj* = *2*
and *tt* ≤ *exec t* <#> *jj*
and *tps′* = *tpsL t xs tt 1* (*λj. sim-move t* ! *j*) (*λj. sim-write t* ! *j*)
shows *execute* (*tmL78 jj*) (*0*, *tps*) *tt* = (*0*, *tps′*)
using *assms*(*6*,*7*)
proof (*induction tt arbitrary*: *tps′*)
case *0*
then show *?case*
using *assms*(*1*−*5*) **by** *simp*
next
case (*Suc tt*)
then have *tt* < *exec t* <#> *jj*
by *simp*
have *execute* (*tmL78 jj*) (*0*, *tps*) (*Suc tt*) = *exe* (*tmL78 jj*) (*execute* (*tmL78 jj*) (*0*, *tps*) *tt*)
by *simp*
also have ... = *exe* (*tmL78 jj*) (*0*, *tpsL t xs tt 1* (*λj. sim-move t* ! *j*) (*λj. sim-write t* ! *j*))
using *Suc* **by** *simp*
also have ... = *sem* (*cmdL8 jj*) (*0*, *tpsL t xs tt 1* (*λj. sim-move t* ! *j*) (*λj. sim-write t* ! *j*))
using *exe-lt-length tmL78-def* **by** *simp*
finally show *?case*
using *assms*(*1*−*5*) *sem-cmdL8-1*[**where** *?i*=*tt*] *Suc* **by** *simp*
qed

lemma *sem-cmdL8-2*:
assumes *jj* < *k*
and *tps* = *tpsL t xs i 1* (*λj. sim-move t* ! *j*) (*λj. sim-write t* ! *j*)
and *length xs* = *k*
and *xs* ! *jj* = (*1*, *Some 0*)
and *i* = *exec t* <#> *jj*
and *sim-move t* ! *jj* = *2*
and *tps′* = *tpsL t* (*xs*[*jj*:=(*1*, *None*)]) (*Suc i*) *1* (*λj. if j* = *jj then 3 else sim-move t* ! *j*) (*λj. sim-write t* ! *j*)
shows *sem* (*cmdL8 jj*) (*0*, *tps*) = (*0*, *tps′*)
proof (*rule semI*[*of 2* ∗ *k* + *3*])
show *proper-command* (*2* ∗ *k* + *3*) (*cmdL8 jj*)
using *turing-command-cmdL8*[*OF assms*(*1*)] *turing-commandD*(*1*) **by** *simp*
show *len*: *length tps* = *2* ∗ *k* + *3*
using *assms*(*2*) **by** *simp*
show *length tps′* = *2* ∗ *k* + *3*
using *assms*(*7*) **by** *simp*

have *i* < *TT*
using *assms exec-pos-less-TT* **by** (*meson Suc-less-eq less-SucI less-trans-Suc*)

define *rs* **where** *rs* = *read tps*
then have *rs* ! *1* ≠ □
using *assms* ‹*i* < *TT*› **by** (*metis not-one-less-zero read-tpsL-1-bounds*(*1*))
then show *fst* (*cmdL8 jj rs*) = *0*
unfolding *cmdL8-def* **by** *simp*

have *rs* ! (*3* + *jj*) = *2*
using *rs-def read-tpsL-3 assms* **by** *simp*
moreover have *enc-nth* (*rs* ! *1*) (*k* + *jj*) = *1*
using *assms rs-def read-tpsL-1-nth-less-2k*[*OF* ‹*i* < *TT*›] **by** *simp*
ultimately have *condition8a rs jj*
using ‹*i* < *TT*› *assms*(*2*) *read-tpsL-1-bounds rs-def* **by** *simp*
then have ∗: *snd* (*cmdL8 jj rs*) =

[(*rs* ! *0*, *Stay*),
 (*enc-upd* (*rs* ! *1*) (*k* + *jj*) *0*, *Right*),
 (*rs* ! *2*, *Stay*)] @
 (*map* (λ*j*. (*if j* = *jj then 3 else rs* ! (*j* + *3*), *Stay*)) [*0*..<*k*]) @
 (*map* (λ*j*. (*rs* ! (*3* + *k* + *j*), *Stay*)) [*0*..<*k*])
unfolding *cmdL8-def* **by** *simp*

show *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *tps′* ! *j* **if** *j* < *2* ∗ *k* + *3* **for** *j*
proof −
 consider *j* = *0* | *j* = *1* | *j* = *2* | *3* ≤ *j* ∧ *j* < *k* + *3* | *k* + *3* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
  using ‹*j* < *2* ∗ *k* + *3*› **by** *linarith*
 then show *?thesis*
 proof (*cases*)
  case *1*
  then have *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL8 jj* (*read tps*) [!] *0*) (*tps* ! *0*)
   by *simp*
  also have ... = *act* (*rs* ! *0*, *Stay*) (*tps* ! *0*)
   using ∗ *rs-def* **by** *simp*
  also have ... = *tps* ! *0*
   using *act-Stay len rs-def* **by** *simp*
  also have ... = *tps′* ! *0*
   using *assms*(*2*,*7*) *tpsL-0* **by** *simp*
  also have ... = *tps′* ! *j*
   using *1* **by** *simp*
  finally show *?thesis* **.**
 next
  case *2*
  then have *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL8 jj* (*read tps*) [!] *1*) (*tps* ! *1*)
   by *simp*
  also have ... = *act* (*enc-upd* (*rs* ! *1*) (*k* + *jj*) *0*, *Right*) (*tps* ! *1*)
   using ∗ *rs-def* **by** *simp*
  also have ... = *tps* ! *1* |:=| (*enc-upd* (*rs* ! *1*) (*k* + *jj*) *0*) |+| *1*
   using *act-Right′ 2 len* **by** *simp*
  also have ... = *tps′* ! *1*
   using *assms zip-cont-enc-upd-None rs-def read-tpsL-1 tpsL-1 zip-cont-def* **by** *simp*
  finally show *?thesis*
   using *2* **by** *simp*
 next
  case *3*
  then have *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL8 jj* (*read tps*) [!] *2*) (*tps* ! *2*)
   by *simp*
  also have ... = *act* (*rs* ! *2*, *Stay*) (*tps* ! *2*)
   using ∗ *rs-def* **by** *simp*
  also have ... = *tps* ! *2*
   using *act-Stay len rs-def* **by** *simp*
  also have ... = *tps′* ! *2*
   using *assms*(*2*,*7*) *tpsL-2* **by** *simp*
  also have ... = *tps′* ! *j*
   using *3* **by** *simp*
  finally show *?thesis* **.**
 next
  case *4*
  show *?thesis*
  proof (*cases j* = *3* + *jj*)
   case *True*
   then have *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*3*, *Stay*) (*tps* ! *j*)
    using ∗ *rs-def threeplus2k-2*[**where** *?a*=(*rs* ! *0*, *Stay*)] *4 diff-add-inverse* **by** (*smt* (*verit*))
   also have ... = *tps′* ! *j*
    using *4 assms*(*2*,*7*) *True act-onesie tpsL-mvs* **by** *simp*
   finally show *?thesis* **.**
  next
   case *False*
   then have *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*rs* ! *j*, *Stay*) (*tps* ! *j*)

using ∗ *rs-def threeplus2k-2*[**where** *?a*=(*rs* ! *0*, *Stay*)] *4* *diff-add-inverse* **by** *auto*
also have ... = *tps′* ! *j*
using *4* *assms*(*2*,*7*) *False act-Stay len rs-def that tpsL-mvs′*
by (*smt* (*verit*) *add.commute le-add-diff-inverse2*)
finally show *?thesis* **.**
qed
next
case *5*
then have *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*rs* ! *j*, *Stay*) (*tps* ! *j*)
using ∗ *rs-def threeplus2k-3*[**where** *?a*=(*rs* ! *0*, *Stay*)] **by** *simp*
also have ... = *tps* ! *j*
using *len act-Stay rs-def that* **by** *simp*
also have ... = *tps′* ! *j*
using *assms*(*2*,*7*) *tpsL-symbs′ 5* **by** *simp*
finally show *?thesis* **.**
qed
qed
qed

lemma *execute-tmL78-2*:
assumes *jj* < *k*
and *tps* = *tpsL t xs 0 1* (*λj. sim-move t* ! *j*) (*λj. sim-write t* ! *j*)
and *length xs* = *k*
and *xs* ! *jj* = (*1*, *Some 0*)
and *sim-move t* ! *jj* = *2*
and *tps′* = *tpsL t* (*xs*[*jj*:=(*1*, *None*)]) (*Suc* (*exec t* <#> *jj*)) *1* (*λj. if j* = *jj then 3 else sim-move t* ! *j*) (*λj. sim-write t* ! *j*)
shows *execute* (*tmL78 jj*) (*0*, *tps*) (*Suc* (*exec t* <#> *jj*)) = (*0*, *tps′*)
using *assms exe-lt-length tmL78-def execute-tmL78-1 sem-cmdL8-2* **by** *simp*

lemma *sem-cmdL8-3*:
assumes *jj* < *k*
and *tps* = *tpsL t xs i 1* (*λj. if j* = *jj then 3 else sim-move t* ! *j*) (*λj. sim-write t* ! *j*)
and *length xs* = *k*
and *xs* ! *jj* = (*1*, *None*)
and *i* = *Suc* (*exec t* <#> *jj*)
and *sim-move t* ! *jj* = *2*
and *tps′* = *tpsL t* (*xs*[*jj*:=(*1*, *Some 1*)]) (*Suc i*) *1* (*λj. sim-move t* ! *j*) (*λj. sim-write t* ! *j*)
shows *sem* (*cmdL8 jj*) (*0*, *tps*) = (*0*, *tps′*)
proof (*rule semI*[*of 2* ∗ *k* + *3*])
show *proper-command* (*2* ∗ *k* + *3*) (*cmdL8 jj*)
using *turing-command-cmdL8*[*OF assms*(*1*)] *turing-commandD*(*1*) **by** *simp*
show *len*: *length tps* = *2* ∗ *k* + *3*
using *assms*(*2*) **by** *simp*
show *length tps′* = *2* ∗ *k* + *3*
using *assms*(*7*) **by** *simp*

have *i* < *TT*
using *assms exec-pos-less-TT sim-move*
by (*metis* (*no-types*, *lifting*) *add-2-eq-Suc′ diff-Suc-1*)
moreover define *rs* **where** *rs* = *read tps*
ultimately have *rs* ! *1* ≠ □
by (*metis* (*no-types*, *lifting*) *assms*(*2*) *not-one-less-zero read-tpsL-1-bounds*(*1*))
then show *fst* (*cmdL8 jj* (*read tps*)) = *0*
using *cmdL8-def rs-def* **by** *simp*

have *rs* ! (*3* + *jj*) = *3*
by (*simp add*: *rs-def assms*(*1*,*2*) *add.commute read-tpsL-3′*)
moreover have *is-code* (*rs* ! *1*)
using *assms rs-def read-tpsL-1-nth-less-2k* ‹*i* < *TT*› *read-tpsL-1-bounds* **by** *simp*
ultimately have *condition7b rs jj*
using ‹*i* < *TT*› *assms*(*2*) *read-tpsL-1-bounds rs-def* **by** *simp*
then have ∗: *snd* (*cmdL8 jj rs*) =

[(*rs* ! *0*, *Stay*),
(*enc-upd* (*rs* ! *1*) (*k* + *jj*) *1*, *Right*),
(*rs* ! *2*, *Stay*)] @
(*map* (λ*j*. (*if j* = *jj then 2 else rs* ! (*j* + *3*), *Stay*)) [*0*..<*k*]) @
(*map* (λ*j*. (*rs* ! (*3* + *k* + *j*), *Stay*)) [*0*..<*k*])
unfolding *cmdL8-def* **by** *simp*

show *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *tps′* ! *j* **if** *j* < *2* ∗ *k* + *3* **for** *j*
proof −
consider *j* = *0* | *j* = *1* | *j* = *2* | *3* ≤ *j* ∧ *j* < *k* + *3* | *k* + *3* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
using ‹*j* < *2* ∗ *k* + *3*› **by** *linarith*
then show *?thesis*
proof (*cases*)
case *1*
then have *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL8 jj* (*read tps*) [!] *0*) (*tps* ! *0*)
by *simp*
also have ... = *act* (*rs* ! *0*, *Stay*) (*tps* ! *0*)
using ∗ *rs-def* **by** *simp*
also have ... = *tps* ! *0*
using *act-Stay len rs-def* **by** *simp*
also have ... = *tps′* ! *0*
using *assms*(*2*,*7*) *tpsL-0* **by** *simp*
also have ... = *tps′* ! *j*
using *1* **by** *simp*
finally show *?thesis* **.**
next
case *2*
then have *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL8 jj* (*read tps*) [!] *1*) (*tps* ! *1*)
by *simp*
also have ... = *act* (*enc-upd* (*rs* ! *1*) (*k* + *jj*) *1*, *Right*) (*tps* ! *1*)
using ∗ *rs-def* **by** *simp*
also have ... = *tps* ! *1* |:=| (*enc-upd* (*rs* ! *1*) (*k* + *jj*) *1*) |+| *1*
using *act-Right′ 2 len* **by** *simp*
also have ... = *tps′* ! *1*
using *assms zip-cont-enc-upd-Some-Right rs-def read-tpsL-1 tpsL-1 zip-cont-def* **by** *simp*
finally show *?thesis*
using *2* **by** *simp*
next
case *3*
then have *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*cmdL8 jj* (*read tps*) [!] *2*) (*tps* ! *2*)
by *simp*
also have ... = *act* (*rs* ! *2*, *Stay*) (*tps* ! *2*)
using ∗ *rs-def* **by** *simp*
also have ... = *tps* ! *2*
using *act-Stay len rs-def* **by** *simp*
also have ... = *tps′* ! *2*
using *assms*(*2*,*7*) *tpsL-2* **by** *simp*
also have ... = *tps′* ! *j*
using *3* **by** *simp*
finally show *?thesis* **.**
next
case *4*
show *?thesis*
proof (*cases j* = *3* + *jj*)
case *True*
then have *act* (*cmdL8 jj* (*read tps*) [!] *j*) (*tps* ! *j*) = *act* (*2*, *Stay*) (*tps* ! *j*)
using ∗ *rs-def threeplus2k-2*[**where** *?a*=(*rs* ! *0*, *Stay*)] *4*
by (*smt* (*verit*, *ccfv-SIG*) *diff-add-inverse*)
also have ... = *tps′* ! *j*
using *4 assms*(*1*,*2*,*6*,*7*) *4 True act-onesie tpsL-mvs* **by** *simp*
finally show *?thesis* **.**
next
case *False*

```
      then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
        using ∗ rs-def threeplus2k-2[where ?a=(rs ! 0, Stay)] 4 diff-add-inverse by auto
      also have ... = tps' ! j
        using 4 assms(2,7) False act-Stay len rs-def that tpsL-mvs'
        by (smt (verit) add.commute le-add-diff-inverse2)
      finally show ?thesis .
    qed
  next
    case 5
    then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
      using ∗ rs-def threeplus2k-3[where ?a=(rs ! 0, Stay)] by simp
    also have ... = tps ! j
      using len act-Stay rs-def that by simp
    also have ... = tps' ! j
      using assms(2,7) tpsL-symbs' 5 by simp
    finally show ?thesis .
  qed
 qed
qed

lemma execute-tmL78-3:
 assumes jj < k
  and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  and length xs = k
  and xs ! jj = (1, Some 0)
  and sim-move t ! jj = 2
  and tps' = tpsL t (xs[jj:=(1, Some 1)]) (Suc (Suc (exec t <#> jj))) 1 (λj. sim-move t ! j) (λj. sim-write t
! j)
 shows execute (tmL78 jj) (0, tps) (Suc (Suc (exec t <#> jj))) = (0, tps')
 using assms exe-lt-length tmL78-def execute-tmL78-2 sem-cmdL8-3 by simp

lemma sem-cmdL8-4:
 assumes jj < k
  and tps = tpsL t xs i 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  and length xs = k
  and xs ! jj = (1, Some 1)
  and i > Suc (exec t <#> jj)
  and i < TT
  and sim-move t ! jj = 2
  and tps' = tpsL t xs (Suc i) 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
 shows sem (cmdL8 jj) (0, tps) = (0, tps')
proof (rule semI[of 2 ∗ k + 3])
 show proper-command (2 ∗ k + 3) (cmdL8 jj)
  using turing-command-cmdL8[OF assms(1)] turing-commandD(1) by simp
 show len: length tps = 2 ∗ k + 3
  using assms(2) by simp
 show length tps' = 2 ∗ k + 3
  using assms(8) by simp

 define rs where rs = read tps
 then have rs ! 1 ≠ □
  using assms by (metis not-one-less-zero read-tpsL-1-bounds(1))
 then show fst (cmdL8 jj rs) = 0
  unfolding cmdL8-def by simp

 have rs ! (3 + jj) = sim-move t ! jj
  using rs-def assms read-tpsL-3 by simp
 moreover have sim-move t ! jj = 2
  using sim-move-def assms(1,7) direction-to-symbol-less sim-move-nth sim-move-nth-else
  by simp
 moreover have enc-nth (rs ! 1) (k + jj) = 0
  using assms rs-def read-tpsL-1-nth-less-2k[OF assms(6), of k + jj] sim-move by simp
 ultimately have condition8d rs jj
```

```
  using assms ‹rs ! 1 ≠ □› by simp
then have *: snd (cmdL8 jj rs) =
  [(rs ! 0, Stay),
   (rs ! 1, Right),
   (rs ! 2, Stay)] @
   (map (λj. (rs ! (j + 3), Stay)) [0..<k]) @
   (map (λj. (rs ! (3 + k + j), Stay)) [0..<k])
  unfolding cmdL8-def by auto

show act (cmdL8 jj (read tps) [!] j) (tps ! j) = tps' ! j if j < 2 * k + 3 for j
proof -
  consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
    using ‹j < 2 * k + 3› by linarith
  then show ?thesis
  proof (cases)
    case 1
    then show ?thesis
      using * act-Stay append.simps(2) assms len nth-Cons-0 rs-def that tpsL-0 by metis
  next
    case 2
    then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (cmdL8 jj (read tps) [!] 1) (tps ! 1)
      by simp
    also have ... = act (rs ! 1, Right) (tps ! 1)
      using * rs-def by simp
    also have ... = tps' ! 1
      using act-Right len rs-def assms tpsL-1 that tpsL-pos-1 2
      by (metis add.commute fst-conv plus-1-eq-Suc)
    also have ... = tps' ! j
      using 2 by simp
    finally show ?thesis .
  next
    case 3
    then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (cmdL8 jj (read tps) [!] 2) (tps ! 2)
      by simp
    also have ... = act (rs ! 2, Stay) (tps ! 2)
      using * rs-def by simp
    also have ... = tps ! 2
      using act-Stay len rs-def by simp
    also have ... = tps' ! 2
      using assms(2,8) tpsL-2 by simp
    also have ... = tps' ! j
      using 3 by simp
    finally show ?thesis .
  next
    case 4
    then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
      using * rs-def threeplus2k-2[where ?a=(rs ! 0, Stay)] by simp
    also have ... = tps ! j
      using len act-Stay rs-def that by simp
    also have ... = tps' ! j
      using assms(2,8) tpsL-mvs' 4 by simp
    finally show ?thesis .
  next
    case 5
    then have act (cmdL8 jj (read tps) [!] j) (tps ! j) = act (rs ! j, Stay) (tps ! j)
      using * rs-def threeplus2k-3[where ?a=(rs ! 0, Stay)] by simp
    also have ... = tps ! j
      using len act-Stay rs-def that by simp
    also have ... = tps' ! j
      using assms(2,8) tpsL-symbs' 5 by simp
    finally show ?thesis .
  qed
qed
```

```
qed

lemma execute-tmL78-4:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (1, Some 0)
    and sim-move t ! jj = 2
    and tt ≥ Suc (Suc (exec t <#> jj))
    and tt ≤ TT
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) tt 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows execute (tmL78 jj) (0, tps) tt = (0, tps')
  using assms(6,7,8)
proof (induction tt arbitrary: tps' rule: nat-induct-at-least)
  case base
  then show ?case
    using assms(1−5) execute-tmL78-3 by simp
next
  case (Suc tt)
  then have tt < TT
    by simp
  let ?xs = xs[jj:=(1, Some 1)]
  let ?tps = tpsL t ?xs tt 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  have execute (tmL78 jj) (0, tps) (Suc tt) = exe (tmL78 jj) (execute (tmL78 jj) (0, tps) tt)
    by simp
  also have ... = exe (tmL78 jj) (0, ?tps)
    using Suc by simp
  also have ... = sem (cmdL8 jj) (0, ?tps)
    using tmL78-def exe-lt-length by simp
  then show ?case
    using sem-cmdL8-4[where ?i=tt] assms Suc by simp
qed

lemma execute-tmL78-5:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (1, Some 0)
    and sim-move t ! jj = 2
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows execute (tmL78 jj) (0, tps) (Suc TT) = (1, tps')
proof −
  have ∗: TT ≥ Suc (Suc (exec t <#> jj))
    using exec-pos-less-TT sim-move assms(1,5)
    by (metis Suc-leI add-2-eq-Suc' add-diff-cancel-left' plus-1-eq-Suc)
  have execute (tmL78 jj) (0, tps) (Suc TT) = exe (tmL78 jj) (execute (tmL78 jj) (0, tps) TT)
    by simp
  also have ... = exe (tmL78 jj) (0, tps')
    using execute-tmL78-4[OF assms(1−5) ∗] assms(6) by simp
  also have ... = sem (cmdL8 jj) (0, tps')
    using tmL78-def exe-lt-length by simp
  finally show ?thesis
    using assms(1,3,6) sem-cmdL8-TT by simp
qed

lemma tmL78-right:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (1, Some 0)
    and sim-move t ! jj = 2
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL78 jj) tps esL78 tps'
```

```
proof
  have len: length esL78 = Suc TT
    using esL78-def by simp
  show execute (tmL78 jj) (0, tps) (length esL78) = (length (tmL78 jj), tps')
    using len execute-tmL78-5 assms tmL78-def by simp
  show fst (execute (tmL78 jj) (0, tps) tt) < length (tmL78 jj)
    if tt < length esL78 for tt
  proof −
    have tt < Suc TT
      using that len by simp
    then consider
        tt ≤ exec t <#> jj
      | tt = Suc (exec t <#> jj)
      | tt = Suc (Suc (exec t <#> jj))
      | tt > Suc (Suc (exec t <#> jj))
      by linarith
    then show ?thesis
    proof (cases)
      case 1
      then show ?thesis
        using assms execute-tmL78-1 tmL78-def by simp
    next
      case 2
      then show ?thesis
        using assms execute-tmL78-2 tmL78-def by simp
    next
      case 3
      then show ?thesis
        using assms execute-tmL78-3 tmL78-def by simp
    next
      case 4
      then show ?thesis
        using assms execute-tmL78-4 tmL78-def ‹tt < Suc TT› by simp
    qed
  qed
  show execute (tmL78 jj) (0, tps) (Suc tt) <#> 0 = fst (esL78 ! tt) ∧
      execute (tmL78 jj) (0, tps) (Suc tt) <#> 1 = snd (esL78 ! tt)
      if tt < length esL78 for tt
  proof −
    have *: Suc tt ≤ Suc TT
      using that esL78-def by simp
    then consider
        Suc tt ≤ exec t <#> jj
      | Suc tt = Suc (exec t <#> jj)
      | Suc tt = Suc (Suc (exec t <#> jj))
      | Suc tt > Suc (Suc (exec t <#> jj)) ∧ Suc tt ≤ TT
      | Suc tt = Suc TT
       by linarith
    then show ?thesis
    proof (cases)
      case 1
      then have esL78 ! tt = (n + 1, Suc tt)
        using nth-map-upt-TT esL78-def by (metis * assms(1) exec-pos-less-TT nat-less-le not-less-eq-eq)
      then show ?thesis
        using execute-tmL78-1[OF assms(1−5), where ?tt=Suc tt] tmL78-def tpsL-pos-0 tpsL-pos-1 * 1
        by simp
    next
      case 2
      then show ?thesis
        using assms execute-tmL78-2[OF assms(1−5)] tmL78-def tpsL-pos-0 tpsL-pos-1 *
        by (metis (no-types, lifting) esL78-def exec-pos-less-TT fst-conv nat.inject nth-map-upt-TT snd-conv)
    next
      case 3
```

```
      then have tt < TT
        by (metis add-2-eq-Suc' assms(1) assms(5) diff-Suc-1 exec-pos-less-TT sim-move)
      then have esL78 ! tt = (n + 1, Suc tt)
        using nth-map-upt-TT esL78-def by auto
      then show ?thesis
        using assms(6) execute-tmL78-3[OF assms(1−5)] tmL78-def tpsL-pos-0 tpsL-pos-1 ∗
        using 3 by simp
    next
      case 4
      then have ∗∗: Suc tt ≥ Suc (Suc (exec t <#> jj))
        by simp
      show ?thesis
        using execute-tmL78-4[OF assms(1−5) ∗∗] tmL78-def tpsL-pos-0 tpsL-pos-1 esL78-def
        by (metis 4 Suc-le-lessD fst-conv nth-map-upt-TT snd-conv)
    next
      case 5
      then have esL78 ! tt = (n + 1, TT)
        using esL78-def by (simp add: nth-append)
      then show ?thesis
        using assms(6) execute-tmL78-5[OF assms(1−5)] tmL78-def tpsL-pos-0 tpsL-pos-1 esL78-def 5
        by simp
    qed
  qed
qed

lemma zip-cont-Stay:
  assumes jj < k
    and length xs = k
    and xs ! jj = (1, Some 0)
    and sim-move t ! jj = 1
  shows zip-cont t xs = zip-cont t (xs[jj:=(1, Some 1)])
proof
  fix i
  let ?xs = xs[jj := (1, Some 1)]
  show zip-cont t xs i = zip-cont t ?xs i
  proof (cases i < TT)
    case True
    then show ?thesis
    proof (rule zip-cont-eqI)
      show ⋀j. j < k ⟹
          (exec (t + fst (xs ! j)) <:> j) i = (exec (t + fst (?xs ! j)) <:> j) i
        using True assms nth-list-update fst-conv by metis
      show ⋀j. j < k ⟹
          (case snd (xs ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) =
          (case snd (?xs ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0)
        using assms sim-move
        by (metis (no-types, lifting) add.commute add.right-neutral add-diff-cancel-right'
          nth-list-update-eq nth-list-update-neq option.simps(5) plus-1-eq-Suc snd-conv)
    qed
  next
    case False
    then show ?thesis
      by (simp add: zip-cont-def)
  qed
qed

lemma tpsL-Stay:
  assumes jj < k
    and tps = tpsL t xs i 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (1, Some 0)
    and sim-move t ! jj = 1
  shows tps = tpsL t (xs[jj:=(1, Some 1)]) i 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
```

```
proof -
  let ?lhs = ((⌊zs⌋, n + 1) #
      (zip-cont t xs, i) #
      ⌈fst (exec (t + 1))⌉ #
      map (onesie ∘ (!) (sim-move t)) [0..<k] @
      map (onesie ∘ (!) (sim-write t)) [0..<k])
  let ?xs = xs[jj:=(1, Some 1)]
  let ?rhs = ((⌊zs⌋, n + 1) #
      (zip-cont t ?xs, i) #
      ⌈fst (exec (t + 1))⌉ #
      map (onesie ∘ (!) (sim-move t)) [0..<k] @
      map (onesie ∘ (!) (sim-write t)) [0..<k])
  have ?lhs = ?rhs
  proof (intro nth-equalityI)
    show length ?lhs = length ?rhs
      by simp
    show ?lhs ! j = ?rhs ! j if j < length ?lhs for j
    proof -
      consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
        using ‹j < length ?lhs› by fastforce
      then show ?thesis
        using zip-cont-Stay assms by (cases) auto
    qed
  qed
  then show ?thesis
    using assms tpsL-def by simp
qed

definition esL48 ≡ esL47 @ esL78

lemma len-esL48: length esL48 = 3 * TT + 3
  using len-esL47 esL48-def esL78-def by simp

lemma tmL48-left:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (0, Some 0)
    and sim-move t ! jj = 0
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL48 jj) tps esL48 tps'
  unfolding tmL48-def esL48-def
  using assms
  by (intro traces-sequential[OF tmL47-left tmL78-nonright[where ?xs=xs[jj:=(1, Some 1)]]]) simp-all

lemma tmL48-right:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (0, Some 0)
    and sim-move t ! jj = 2
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL48 jj) tps esL48 tps'
  unfolding tmL48-def esL48-def
  using assms
  by (intro traces-sequential[OF tmL47-nonleft tmL78-right[where ?xs=xs[jj:=(1, Some 0)]]]) simp-all

lemma tmL48-stay:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (0, Some 0)
    and sim-move t ! jj = 1
```

```
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL48 jj) tps esL48 tps'
proof –
  let ?xs = xs[jj:=(1, Some 0)]
  let ?tps = tpsL t ?xs TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  have traces (tmL48 jj) tps esL48 ?tps
    unfolding tmL48-def esL48-def
    using assms
    by (intro traces-sequential[OF tmL47-nonleft tmL78-nonright[where ?xs=?xs]]) simp-all
  then show ?thesis
    using tpsL-Stay[where ?xs=?xs] assms by simp
qed

lemma tmL48:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (0, Some 0)
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL48 jj) tps esL48 tps'
proof –
  consider sim-move t ! jj = 0 | sim-move t ! jj = 1 | sim-move t ! jj = 2
    using direction-to-symbol-less sim-move-def assms(1) sim-move-nth sim-move-nth-else
    by (metis (no-types, lifting) One-nat-def Suc-1 less-Suc-eq less-nat-zero-code numeral-3-eq-3)
  then show ?thesis
    using assms tmL48-left tmL48-right tmL48-stay
    by (cases) simp-all
qed

definition esL49 ≡ esL48 @ map (λi. (n + 1, i)) (rev [0..<TT]) @ [(n + 1, 0)]

lemma len-esL49: length esL49 = 4 * TT + 4
  using len-esL48 esL49-def by simp

lemma tmL49:
  assumes jj < k
    and tps = tpsL t xs 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
    and length xs = k
    and xs ! jj = (0, Some 0)
    and tps' = tpsL t (xs[jj:=(1, Some 1)]) 0 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  shows traces (tmL49 jj) tps esL49 tps'
  unfolding tmL49-def esL49-def
proof (rule traces-sequential)
  let ?tps = tpsL t (xs[jj:=(1, Some 1)]) TT 1 (λj. sim-move t ! j) (λj. sim-write t ! j)
  show traces (tmL48 jj) tps esL48 ?tps
    using assms tmL48 by simp
  show traces tm-left-until1 ?tps (map (Pair (n + 1)) (rev [0..<Suc (fmt n)]) @ [(n + 1, 0)]) tps'
  proof (rule traces-tm-left-until-1I)
    show 1 < length ?tps
      by simp
    show begin-tape {y. y < G ^ (2 * k + 2) + 2 ∧ 1 < y ∧ dec y ! (2 * k + 1) = 1} (?tps ! 1)
      using tpsL-1 begin-tape-zip-cont by simp
    show map (Pair (n + 1)) (rev [0..<Suc (fmt n)]) @ [(n + 1, 0)] =
        map (Pair (?tps :#: 0)) (rev [0..< ?tps :#: 1]) @ [(?tps :#: 0, 0)]
      using tpsL-pos-0 tpsL-pos-1 by presburger
    show tps' = ?tps [1 := ?tps ! 1 |#=| 0]
      using assms tpsL-def by simp
  qed
qed

definition xs49 :: nat ⇒ (nat × nat option) list where
  xs49 j ≡ replicate j (1, Some 1) @ replicate (k − j) (0, Some 0)
```

lemma *length-xs49*: $j \leq k \Longrightarrow$ *length* (*xs49 j*) = *k*
  using *xs49-def* **by** *simp*

lemma *xs49-less*:
  assumes $j \leq k$ **and** $i < j$
  shows *xs49 j* ! *i* = (*1*, *Some 1*)
  unfolding *xs49-def* **using** *assms* **by** (*simp add*: *nth-append*)

lemma *xs49-ge*:
  assumes $j \leq k$ **and** $i \geq j$ **and** $i < k$
  shows *xs49 j* ! *i* = (*0*, *Some 0*)
  unfolding *xs49-def* **using** *assms* **by** (*simp add*: *nth-append*)

lemma *xs49-upd*:
  assumes $j < k$
  shows *xs49* (*Suc j*) = (*xs49 j*)[*j* := (*1*, *Some 1*)]
    (**is** *?lhs* = *?rhs*)
proof (*rule nth-equalityI*)
  show *length ?lhs* = *length ?rhs*
    by (*simp add*: *Suc-leI assms length-xs49 less-imp-le-nat*)
  show $\bigwedge i.$ *i* < *length ?lhs* $\Longrightarrow$ *?lhs* ! *i* = *?rhs* ! *i*
    using *length-xs49 assms xs49-ge xs49-less*
    by (*metis less-Suc-eq less-or-eq-imp-le not-less nth-list-update*)
qed

lemma *tmL49-upt*:
  assumes $j \leq k$
    and *tps'* = *tpsL t* (*xs49 j*) *0 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
  shows *traces* (*tmL49-upt j*) (*tpsL4 t*) (*concat* (*replicate j esL49*)) *tps'*
  using *assms*
proof (*induction j arbitrary*: *tps'*)
  case *0*
  then show *?case*
    using *xs49-def tpsL4-def assms* **by** *auto*
next
  case (*Suc j*)
  then have *j* < *k*
    by *simp*
  let *?tps* = *tpsL t* (*xs49 j*) *0 1* (λ*j*. *sim-move t* ! *j*) (λ*j*. *sim-write t* ! *j*)
  have *tmL49-upt* (*Suc j*) = *tmL49-upt j* ;; *tmL49 j*
    by *simp*
  moreover have *concat* (*replicate* (*Suc j*) *esL49*) = *concat* (*replicate j esL49*) @ *esL49*
    by (*smt* (*verit*) *append.assoc append-replicate-commute append-same-eq concat.simps*(*2*) *concat-append*
      replicate.simps(*2*))
  moreover have *traces* (*tmL49-upt j* ;; *tmL49 j*) (*tpsL4 t*) (*concat* (*replicate j esL49*) @ *esL49*) *tps'*
  proof (*rule traces-sequential*)
    show *traces* (*tmL49-upt j*) (*tpsL4 t*) (*concat* (*replicate j esL49*)) *?tps*
      using *Suc* **by** *simp*
    show *traces* (*tmL49 j*) *?tps esL49 tps'*
      using *tmL49*[*OF* ‹*j* < *k*›, **where** *?tps*=*?tps*] *length-xs49 xs49-upd Suc* ‹*j* < *k*› *xs49-ge*
      by *simp*
  qed
  ultimately show *?case*
    by *simp*
qed

definition *esL49-upt* ≡ *concat* (*replicate k esL49*)

lemma *length-concat-replicate*: *length* (*concat* (*replicate m xs*)) = *m* ∗ *length xs*
  by (*induction m*) *simp-all*

lemma *len-esL49-upt*: *length esL49-upt* = *k* ∗ (*4* ∗ *TT* + *4*)
  using *len-esL49 esL49-upt-def length-concat-replicate*[*of k esL49*] **by** *simp*

corollary *tmL49-upt'*:
assumes $tps' = tpsL\ t\ (xs49\ k)\ 0\ 1\ (\lambda j.\ sim\text{-}move\ t\ !\ j)\ (\lambda j.\ sim\text{-}write\ t\ !\ j)$
shows *traces* (*tmL49-upt k*) (*tpsL4 t*) *esL49-upt tps'*
using *tmL49-upt*[*of k*] *assms esL49-upt-def* **by** *simp*

definition *esL9 t* ≡ *esL4 t* @ *esL49-upt*

lemma *len-esL9*: $length\ (esL9\ t) = k * (4 * TT + 4) + t + 2 * TT + 4$
using *len-esL4 len-esL49-upt esL9-def* **by** *simp*

lemma *xs49-k*: *xs49 k* = *replicate k* (*1*, *Some 1*)
using *xs49-def* **by** *simp*

definition *tpsL9 t* ≡ *tpsL*
t
(*replicate k* (*1*, *Some 1*))
0
1
(λj. *sim-move t* ! *j*)
(λj. *sim-write t* ! *j*)

lemma *tmL9*:
assumes $t < TT$
shows *traces tmL9* (*tpsL0 t*) (*esL9 t*) (*tpsL9 t*)
unfolding *tmL9-def esL9-def*
using *assms tmL4 tmL49-upt'*
by (*intro traces-sequential*) (*auto simp add*: *tpsL9-def xs49-k*)

definition *esL10 t* ≡ *esL9 t* @ *esC t*

lemma *len-esL10*: $length\ (esL10\ t) = k * (4 * TT + 4) + 2 * t + 2 * TT + 5$
using *len-esL9 len-esL9 esL10-def esC-def* **by** *simp*

definition *tpsL10 t* ≡ *tpsL*
t
(*replicate k* (*1*, *Some 1*))
t
1
(λj. *sim-move t* ! *j*)
(λj. *sim-write t* ! *j*)

lemma *tmL10*:
assumes $t < TT$
shows *traces tmL10* (*tpsL0 t*) (*esL10 t*) (*tpsL10 t*)
unfolding *tmL10-def esL10-def*
proof (*rule traces-sequential*[*OF tmL9*[*OF assms*]])
have $t \leq TT$
using *assms* **by** *simp*
then show *traces tmC* (*tpsL9 t*) (*esC t*) (*tpsL10 t*)
using *tmC-general tpsL9-def tpsL10-def* **by** *simp*
qed

definition *tpsL11 t* ≡ *tpsL*
(*Suc t*)
(*replicate k* (*0*, *Some 0*))
t
0
(λj. *sim-move t* ! *j*)
(λj. *sim-write t* ! *j*)

lemma *enc-upd-2k*:
assumes *dec n* = (*map f* [*0*..<*k*] @ *map h* [*0*..<*k*] @ [*a*, *b*])

```
  shows enc-upd n (2 * k) 1 = enc (map f [0..<k] @ map h [0..<k] @ [1, b])
  using assms enc-upd-def by (metis append-assoc list-update-length nth-list-update-neq twok-nth)

lemma enc-upd-zip-cont:
  assumes t < TT
    and xs1 = replicate k (1, Some 1)
    and xs0 = (replicate k (0, Some 0))
  shows enc-upd (zip-cont t xs1 t) (2 * k) 1 = zip-cont (Suc t) xs0 t
proof -
  let ?n = zip-cont t xs1 t
  have xs1: fst (xs1 ! j) = 1 snd (xs1 ! j) = Some 1 if j < k for j
    using assms(2) that by simp-all
  have zip-cont t xs1 t = enc
      (map (λj. (exec (t + fst (xs1 ! j)) <:> j) t) [0..<k] @
       map (λj. case snd (xs1 ! j) of None ⇒ 0 | Some d ⇒ if t = exec (t + d) <#> j then 1 else 0) [0..<k] @
       [0,
        if t = 0 then 1 else 0])
    using zip-cont-def assms(1) by simp
  also have ... = enc
      (map (λj. (exec (t + 1) <:> j) t) [0..<k] @
       map (λj. case Some 1 of None ⇒ 0 | Some d ⇒ if t = exec (t + d) <#> j then 1 else 0) [0..<k] @
       [0,
        if t = 0 then 1 else 0])
    using xs1 by (smt (verit) atLeastLessThan-iff map-eq-conv set-upt)
  finally have 1: zip-cont t xs1 t = enc
      (map (λj. (exec (Suc t) <:> j) t) [0..<k] @
       map (λj. if t = exec (Suc t) <#> j then 1 else 0) [0..<k] @
       [0,
        if t = 0 then 1 else 0])
        (is - = enc ?ys)
    by simp

  have xs0: fst (xs0 ! j) = 0 snd (xs0 ! j) = Some 0 if j < k for j
    using assms(3) that by simp-all
  have zip-cont (Suc t) xs0 t = enc
      (map (λj. (exec (Suc t + fst (xs0 ! j)) <:> j) t) [0..<k] @
       map (λj. case snd (xs0 ! j) of None ⇒ 0 | Some d ⇒ if t = exec (Suc t + d) <#> j then 1 else 0) [0..<k]
@
       [1,
        if t = 0 then 1 else 0])
    using zip-cont-def assms(1) by simp
  also have ... = enc
      (map (λj. (exec (Suc t) <:> j) t) [0..<k] @
       map (λj. case Some 0 of None ⇒ 0 | Some d ⇒ if t = exec (Suc t + d) <#> j then 1 else 0) [0..<k] @
       [1,
        if t = 0 then 1 else 0])
    using xs0 by (smt (verit, ccfv-SIG) add.right-neutral atLeastLessThan-iff map-eq-conv set-upt)
  finally have 2: zip-cont (Suc t) xs0 t = enc
      (map (λj. (exec (Suc t) <:> j) t) [0..<k] @
       map (λj. if t = exec (Suc t) <#> j then 1 else 0) [0..<k] @
       [1,
        if t = 0 then 1 else 0])
        (is - = enc ?zs)
    by simp
  moreover have ?zs = ?ys [2 * k := 1]
    by (smt (verit) Suc-1 append-assoc diff-zero length-append length-map length-upt list-update-length mult-Suc
nat-mult-1)
  moreover have ?ys = dec ?n
    using dec-zip-cont assms by simp
  ultimately show ?thesis
    using enc-upd-def 1 by presburger
qed
```

```
lemma enc-upd-zip-cont-upd:
  assumes t < TT
    and xs1 = replicate k (1, Some 1)
    and xs0 = (replicate k (0, Some 0))
  shows (zip-cont t xs1) (t:=enc-upd (zip-cont t xs1 t) (2 ∗ k) 1) = zip-cont (Suc t) xs0
proof
  fix i
  consider i = t | i ≠ t ∧ i < TT | i ≥ TT
    using assms(1) by linarith
  then show ((zip-cont t xs1)(t := enc-upd (zip-cont t xs1 t) (2 ∗ k) 1)) i = zip-cont (Suc t) xs0 i
  proof (cases)
    case 1
    then show ?thesis
      using enc-upd-zip-cont assms by simp
  next
    case 2
    then have i ≠ t i < TT
      by simp-all
    have xs1: fst (xs1 ! j) = 1 snd (xs1 ! j) = Some 1 if j < k for j
      using assms(2) that by simp-all
    have zip-cont t xs1 i = enc
      (map (λj. (exec (t + fst (xs1 ! j)) <:> j) i) [0..<k] @
       map (λj. case snd (xs1 ! j) of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k] @
       [if i < t then 1 else 0,
        if i = 0 then 1 else 0])
      using zip-cont-def assms(1) ‹i < TT› by simp
    also have ... = enc
      (map (λj. (exec (t + 1) <:> j) i) [0..<k] @
       map (λj. case Some 1 of None ⇒ 0 | Some d ⇒ if i = exec (t + d) <#> j then 1 else 0) [0..<k] @
       [if i < t then 1 else 0,
        if i = 0 then 1 else 0])
      using xs1 by (smt (verit) atLeastLessThan-iff map-eq-conv set-upt)
    finally have 1: zip-cont t xs1 i = enc
      (map (λj. (exec (Suc t) <:> j) i) [0..<k] @
       map (λj. if i = exec (Suc t) <#> j then 1 else 0) [0..<k] @
       [if i < t then 1 else 0,
        if i = 0 then 1 else 0])
      by simp

    have xs0: fst (xs0 ! j) = 0 snd (xs0 ! j) = Some 0 if j < k for j
      using assms(3) that by simp-all
    have zip-cont (Suc t) xs0 i = enc
      (map (λj. (exec (Suc t + fst (xs0 ! j)) <:> j) i) [0..<k] @
       map (λj. case snd (xs0 ! j) of None ⇒ 0 | Some d ⇒ if i = exec (Suc t + d) <#> j then 1 else 0) [0..<k]
@
       [if i < Suc t then 1 else 0,
        if i = 0 then 1 else 0])
      using zip-cont-def[of Suc t xs0 i] ‹i < TT› assms(1) by simp
    also have ... = enc
      (map (λj. (exec (Suc t) <:> j) i) [0..<k] @
       map (λj. case Some 0 of None ⇒ 0 | Some d ⇒ if i = exec (Suc t + d) <#> j then 1 else 0) [0..<k] @
       [if i < Suc t then 1 else 0,
        if i = 0 then 1 else 0])
      using xs0 by (smt (verit, ccfv-SIG) add.right-neutral atLeastLessThan-iff map-eq-conv set-upt)
    finally have 2: zip-cont (Suc t) xs0 i = enc
      (map (λj. (exec (Suc t) <:> j) i) [0..<k] @
       map (λj. if i = exec (Suc t) <#> j then 1 else 0) [0..<k] @
       [if i < t then 1 else 0,
        if i = 0 then 1 else 0])
      using ‹i ≠ t› by simp
    then show ?thesis
      using 1 ‹i ≠ t› by simp
  next
```

```
    case 3
    then show ?thesis
      using zip-cont-def assms(1) by simp
  qed
qed

lemma sem-cmdL11:
  assumes t < TT
  shows sem cmdL11 (0, tpsL10 t) = (1, tpsL11 t)
proof (rule semI[of 2 ∗ k + 3])
  show proper-command (2 ∗ k + 3) cmdL11
    using cmdL11-def by simp
  show len: length (tpsL10 t) = 2 ∗ k + 3 length (tpsL11 t) = 2 ∗ k + 3
    using tpsL10-def tpsL11-def by simp-all
  show fst (cmdL11 (read (tpsL10 t))) = 1
    using cmdL11-def by simp
  let ?tps = tpsL10 t
  let ?xs = replicate k (1::nat, Some 1::nat option)
  have tps1: ?tps ! 1 = (zip-cont t ?xs, t)
    using tpsL-1 tpsL10-def by simp
  have tps1': tpsL11 t ! 1 = (zip-cont (Suc t) (replicate k (0, Some 0)), t)
    using tpsL-1 tpsL11-def by simp
  let ?rs = read ?tps
  have is-code (?rs ! 1)
    using tpsL10-def assms read-tpsL-1-bounds by simp
  have rs1: ?rs ! 1 = zip-cont t ?xs t
    using tps1 read-def tpsL-def tpsL10-def by force
  show  act (cmdL11 ?rs [!] j) (?tps ! j) = tpsL11 t ! j
      if j < 2 ∗ k + 3 for j
    proof −
    consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 ∗ k + 3
      using ‹j < 2 ∗ k + 3› by linarith
    then show ?thesis
    proof (cases)
      case 1
      then show ?thesis
        using tpsL10-def tpsL11-def read-tpsL-0 cmdL11-def act-Stay len(1) that tpsL-0
        by (smt (verit) append-Cons nth-Cons-0 prod.sel(2))
    next
      case 2
      then have act (cmdL11 ?rs [!] j) (?tps ! j) = act (cmdL11 ?rs [!] 1) (?tps ! 1)
        by simp
      also have ... = act (enc-upd (?rs ! 1) (2 ∗ k) 1, Stay) (?tps ! 1)
        using cmdL11-def ‹is-code (?rs ! 1)› by simp
      also have ... = (?tps ! 1) |:=| enc-upd (?rs ! 1) (2 ∗ k) 1
        by (simp add: act-Stay tps1 2 act-Stay' len(1) that)
      also have ... = tpsL11 t ! 1
        using enc-upd-zip-cont-upd rs1 tps1' tps1 assms by simp
      finally show ?thesis
        using 2 by simp
    next
      case 3
      then have act (cmdL11 ?rs [!] j) ((?tps) ! j) = act (cmdL11 ?rs [!] 2) (?tps ! 2)
        by simp
      also have ... = act (?rs ! 2, Stay) (?tps ! 2)
        using cmdL11-def by simp
      also have ... = ?tps ! 2
        using act-Stay len by simp
      also have ... = (tpsL11 t) ! 2
        using tpsL-2 tpsL11-def tpsL10-def by simp
      also have ... = (tpsL11 t) ! j
        using 3 by simp
      finally show ?thesis .
```

```
    next
      case 4
      then have act (cmdL11 ?rs [!] j) (?tps ! j) = act (?rs ! j, Stay) (?tps ! j)
        using cmdL11-def threeplus2k-2[where ?a=(?rs ! 0, Stay)] by simp
      also have ... = (tpsL10 t) ! j
        using len act-Stay that by simp
      also have ... = (tpsL11 t) ! j
        using tpsL11-def tpsL10-def tpsL-mvs' 4 by simp
      finally show ?thesis .
    next
      case 5
      then have act (cmdL11 ?rs [!] j) (?tps ! j) = act (?rs ! j, Stay) (?tps ! j)
        using cmdL11-def threeplus2k-3[where ?a=(?rs ! 0, Stay)] by simp
      also have ... = (tpsL10 t) ! j
        using len act-Stay that by simp
      also have ... = (tpsL11 t) ! j
        using tpsL11-def tpsL10-def tpsL-symbs' 5 by simp
      finally show ?thesis .
    qed
  qed
qed

definition esL11 t ≡ esL10 t @ [(n + 1, t)]

lemma len-esL11: length (esL11 t) = k * (4 * TT + 4) + 2 * t + 2 * TT + 6
  using len-esL10 esL11-def by simp

lemma tmL11:
  assumes t < TT
  shows traces tmL11 (tpsL0 t) (esL11 t) (tpsL11 t)
  unfolding tmL11-def esL11-def
proof (rule traces-sequential[OF tmL10[OF assms]])
  let ?cmd = [cmdL11]
  let ?es = [(n + 1, t)]
  show traces ?cmd (tpsL10 t) ?es (tpsL11 t)
  proof
    show 1: execute ?cmd (0, tpsL10 t) (length ?es) = (length ?cmd, tpsL11 t)
    proof -
      have execute ?cmd (0, tpsL10 t) (length ?es) = exe ?cmd (0, tpsL10 t)
        by simp
      also have ... = sem cmdL11 (0, tpsL10 t)
        using exe-lt-length cmdL11-def by simp
      finally show ?thesis
        using sem-cmdL11[OF assms] by simp
    qed
    show ⋀i. i < length [(n + 1, t)] ⟹
          fst (execute ?cmd (0, tpsL10 t) i) < length ?cmd
      by simp
    show ⋀i. i < length [(n + 1, t)] ⟹
          execute ?cmd (0, tpsL10 t) (Suc i) <#> 0 = fst (?es ! i) ∧
          execute ?cmd (0, tpsL10 t) (Suc i) <#> 1 = snd (?es ! i)
      using 1 tpsL11-def tpsL-pos-0 tpsL-pos-1
        by (metis One-nat-def add.commute fst-conv less-Suc0 list.size(3) list.size(4) nth-Cons-0 plus-1-eq-Suc
snd-conv)
  qed
qed

definition esL12 t ≡ esL11 t @ map (λi. (n + 1, i)) (rev [0..<t]) @ [(n + 1, 0)]

lemma len-esL12: length (esL12 t) = k * (4 * TT + 4) + 3 * t + 2 * TT + 7
  using len-esL11 esL12-def by simp

definition tpsL12 t ≡ tpsL
```

```
  (Suc t)
  (replicate k (0, Some 0))
  0
  0
  (λj. sim-move t ! j)
  (λj. sim-write t ! j)

lemma tmL12:
  assumes t < TT
  shows traces tmL12 (tpsL0 t) (esL12 t) (tpsL12 t)
  unfolding tmL12-def esL12-def
proof (rule traces-sequential[OF tmL11[OF assms]])
  show traces tm-left-until1 (tpsL11 t) (map (Pair (n + 1)) (rev [0..<t]) @ [(n + 1, 0)]) (tpsL12 t)
  proof (rule traces-tm-left-until-1I)
    show 1 < length (tpsL11 t)
      using tpsL11-def by simp
    show begin-tape {y. y < G ^ (2 * k + 2) + 2 ∧ 1 < y ∧ dec y ! (2 * k + 1) = 1} (tpsL11 t ! 1)
      using tpsL-1 begin-tape-zip-cont tpsL11-def by simp
    show map (Pair (n + 1)) (rev [0..<t]) @ [(n + 1, 0)] =
      map (Pair (tpsL11 t :#: 0)) (rev [0..<tpsL11 t :#: 1]) @ [(tpsL11 t :#: 0, 0)]
      using tpsL-pos-0 tpsL-pos-1 tpsL11-def by simp
    show tpsL12 t = (tpsL11 t)[1 := tpsL11 t ! 1 |#=| 0]
      using tpsL12-def tpsL11-def tpsL-def by simp
  qed
qed

definition tpsL13 t ≡ tpsL
    (Suc t)
    (replicate k (0, Some 0))
    0
    0
    (λj. 0)
    (λj. 0)

definition esL13 t ≡ esL12 t @ [(n + 1, 0)]

lemma len-esL13: length (esL13 t) = k * (4 * TT + 4) + 3 * t + 2 * TT + 8
  using len-esL12 esL13-def by simp

lemma tmL13:
  assumes t < TT
  shows traces tmL13 (tpsL0 t) (esL13 t) (tpsL13 t)
  unfolding tmL13-def esL13-def
proof (rule traces-sequential[OF tmL12[OF assms]])
  show traces (tm-write-many {3..<2 * k + 3} 0) (tpsL12 t) [(n + 1, 0)] (tpsL13 t)
  proof (rule traces-tm-write-manyI[where ?k=2*k+3])
    show 0 ∉ {3..<2 * k + 3}
      by simp
    show ∀j∈{3..<2 * k + 3}. j < 2 * k + 3
      by simp
    show 2 ≤ 2 * k + 3
      by simp
    show length (tpsL12 t) = 2 * k + 3 length (tpsL13 t) = 2 * k + 3
      using tpsL12-def tpsL13-def length-tpsL by simp-all
    show tpsL13 t ! j = tpsL12 t ! j |:=| 0 if j ∈ {3..<2 * k + 3} for j
    proof (cases j < k + 3)
      case True
      then have 3 ≤ j ∧ j < k + 3
        using that by simp
      then show ?thesis
        using tpsL13-def tpsL12-def tpsL-mvs' onesie-write by simp
    next
      case False
```

```
      then have k + 3 ≤ j ∧ j < 2 ∗ k + 3
        using that by simp
      then show ?thesis
        using tpsL13-def tpsL12-def tpsL-symbs′ onesie-write by simp
    qed
    show tpsL13 t ! j = tpsL12 t ! j if j < 2 ∗ k + 3 j ∉ {3..<2 ∗ k + 3} for j
    proof −
      from that have j < 3
        by simp
      then show ?thesis
        using tpsL13-def tpsL12-def tpsL-def less-Suc-eq numeral-3-eq-3 by auto
    qed
    show [(n + 1, 0)] = [(tpsL12 t :#: 0, tpsL12 t :#: 1)]
      using tpsL12-def tpsL-pos-0 tpsL-pos-1 by simp
  qed
qed

corollary tmL13′:
  assumes t < TT
  shows traces tmL13 (tpsC1 t) (esL13 t) (tpsL13 t)
  using tmL13 tpsC1-def tpsL0-def assms by simp

definition esLoop-while t ≡
  esC t @ [(tpsC1 t :#: 0, tpsC1 t :#: 1)] @ esL13 t @ [(tpsL13 t :#: 0, tpsL13 t :#: 1)]

definition esLoop-break ≡ (esC TT) @ [(tpsC1 TT :#: 0, tpsC1 TT :#: 1)]

lemma len-esLoop-while: length (esLoop-while t) = k ∗ (4 ∗ TT + 4) + 4 ∗ t + 2 ∗ TT + 11
  using len-esL13 esC-def esLoop-while-def by simp

lemma tmLoop-while:
  assumes t < TT
  shows trace tmLoop (0, tpsC0 t) (esLoop-while t) (0, tpsL13 t)
  unfolding tmLoop-def
proof (rule tm-loop-sem-true-tracesI[OF tmC tmL13′])
  show t ≤ TT and t < TT
    using assms by simp-all
  show 0 < read (tpsC1 t) ! 1
    using tpsC1-def read-tpsL-1-bounds(1) assms by (metis gr0I not-one-less-zero)
  show esLoop-while t =
      esC t @ [(tpsC1 t :#: 0, tpsC1 t :#: 1)] @ esL13 t @ [(tpsL13 t :#: 0, tpsL13 t :#: 1)]
    using esLoop-while-def by simp
qed

lemma tmLoop-while-end:
  trace tmLoop (0, tpsC0 0) (concat (map esLoop-while [0..<TT])) (0, tpsC0 TT)
proof (rule tm-loop-trace-simple)
  have tpsL13 t = tpsC0 (Suc t) if t < TT for t
    using tpsL13-def tpsC0-def by simp
  then show trace tmLoop (0, tpsC0 i) (esLoop-while i) (0, tpsC0 (Suc i)) if i < TT for i
    using tmLoop-while that by simp
qed

lemma len-esLoop-break: length esLoop-break = TT + 2
  using esLoop-break-def esC-def by simp

lemma tmLoop-break: traces tmLoop (tpsC0 TT) esLoop-break (tpsC1 TT)
  unfolding tmLoop-def
proof (rule tm-loop-sem-false-traces[OF tmC])
  show TT ≤ TT
    by simp
  show ¬ 0 < read (tpsC1 TT) ! 1
    using read-tpsL-1 tpsC1-def by simp
```

show *esLoop-break* = *esC* (*TT*) @ [(*tpsC1 TT* :#: *0*, *tpsC1 TT* :#: *1*)]
  using *esLoop-break-def* **by** *simp*
qed

definition *esLoop* ≡ *concat* (*map esLoop-while* [*0*..<*TT*]) @ *esLoop-break*

lemma *len-esLoop1*: $u \leq TT \Longrightarrow$ *length* (*concat* (*map esLoop-while* [*0*..<*u*])) $\leq u * (k * (4 * TT + 4) + 4 * TT + 2 * TT + 11)$
  using *len-esLoop-while* **by** (*induction u*) *simp-all*

lemma *len-esLoop2*: *length* (*concat* (*map esLoop-while* [*0*..<*TT*])) $\leq TT * (k * (4 * TT + 4) + 4 * TT + 2 * TT + 11)$
  using *len-esLoop1*[*of TT*] **by** *simp*

lemma *len-esLoop3*: *length esLoop* $\leq TT * (k * (4 * TT + 4) + 4 * TT + 2 * TT + 11) + TT + 2$
  using *len-esLoop2 esC-def esLoop-def esLoop-break-def* **by** *simp*

lemma *len-esLoop*: *length esLoop* $\leq 28 * k * TT * TT$
proof −
  have *length esLoop* $\leq TT * (k * (4 * TT + 4) + 4 * TT + 2 * TT + 11) + TT + 2$
    using *len-esLoop3* **.**
  also have ... $= TT * (k * (4 * TT + 4) + 6 * TT + 11) + TT + 2$
    by *simp*
  also have ... $\leq TT * (k * (4 * TT + 4) + 6 * TT + 11) + 3 * TT$
    by *simp*
  also have ... $= TT * k * (4 * TT + 4) + TT * 6 * TT + TT * 11 + 3 * TT$
    by *algebra*
  also have ... $= TT * k * (4 * TT + 4) + TT * 6 * TT + 14 * TT$
    by *simp*
  also have ... $= k * 4 * TT * TT + k * 4 * TT + 6 * TT * TT + 14 * TT$
    by *algebra*
  also have ... $\leq k * 4 * TT * TT + k * 4 * TT * TT + 6 * TT * TT + 14 * TT * TT$
    by *simp*
  also have ... $= (k * 4 + k * 4 + 6 + 14) * TT * TT$
    by *algebra*
  also have ... $= (k * 8 + 20) * TT * TT$
    by *algebra*
  also have ... $\leq 28 * k * TT * TT$
  proof −
    have $k * 8 + 20 \leq 28 * k$
      using *k-ge-2* **by** *simp*
    then show *?thesis*
      by (*meson mult-le-mono1*)
  qed
  finally show *?thesis*
    by *simp*
qed

lemma *tmLoop*: *traces tmLoop* (*tpsC0 0*) *esLoop* (*tpsC1 TT*)
  unfolding *esLoop-def* **using** *traces-additive*[*OF tmLoop-while-end tmLoop-break*] **.**

lemma *tps9-tpsC0*: *tps9* = *tpsC0 0*
  using *tps9-def tpsC0-def tps9-tpsL* **by** *simp*

definition *es10* ≡ *es9* @ *esLoop*

lemma *len-es10*: *length es10* $\leq$ *length* (*es-fmt n*) $+ 40 * k * TT * TT$
proof −
  have *length es10* $\leq$ *length* (*es-fmt n*) $+ 2 * TT + 2 * n + 8 + 28 * k * TT * TT$
    using *len-es9 len-esLoop es10-def* **by** *simp*
  also have ... $\leq$ *length* (*es-fmt n*) $+ 2 * TT + 2 * TT + 8 + 28 * k * TT * TT$
  proof −
    have $2 * n \leq 2 * TT$

using *fmt-ge-n Suc-mult-le-cancel1 le-SucI numeral-2-eq-2* **by** *metis*
then show *?thesis*
by *simp*
qed
also have ... $\leq$ *length* (*es-fmt n*) + *12* $*$ *TT* $*$ *TT* + *28* $*$ *k* $*$ *TT* $*$ *TT*
by *simp*
also have ... $\leq$ *length* (*es-fmt n*) + *12* $*$ *k* $*$ *TT* $*$ *TT* + *28* $*$ *k* $*$ *TT* $*$ *TT*
proof −
have *12* $\leq$ *12* $*$ *k*
using *k-ge-2* **by** *simp*
then have *12* $*$ *TT* $*$ *TT* $\leq$ *12* $*$ *k* $*$ *TT* $*$ *TT*
using *mult-le-mono1* **by** *presburger*
then show *?thesis*
by *simp*
qed
also have ... = *length* (*es-fmt n*) + *40* $*$ *k* $*$ *TT* $*$ *TT*
by *linarith*
finally show *?thesis* **.**
qed

lemma *tm10*: *traces tm10 tps0 es10* (*tpsC1 TT*)
unfolding *tm10-def es10-def*
using *traces-sequential*[*OF tm9*] *tps9-tpsC0 tmLoop*
by *simp*

Cleaning up the output

abbreviation *tps10* ≡ *tpsC1 TT*

definition *es11* ≡ *es10* @ *map* (λ*i*. (*n* + *1*, *i*)) (*rev* [*0*..<*TT*]) @ [(*n* + *1*, *0*)]

lemma *len-es11*: *length es11* $\leq$ *length* (*es-fmt n*) + *40* $*$ *k* $*$ *TT* $*$ *TT* + *Suc TT*
using *es11-def len-es10* **by** *simp*

definition *tps11* ≡ *tps10*[*1* := *ltransplant* (*tps10* ! *1*) (*tps10* ! *1*) *ec1 TT*]

lemma *tm11*: *traces tm11 tps0 es11 tps11*
unfolding *tm11-def es11-def*
proof (*rule traces-sequential*[*OF tm10*])
show *traces*
(*tm-ltrans-until 1 1 StartSym ec1*)
(*tpsC1* (*Suc* (*fmt n*)))
(*map* (*Pair* (*n* + *1*)) (*rev* [*0*..<*Suc* (*fmt n*)]) @ [(*n* + *1*, *0*)]) *tps11*
proof (*rule traces-tm-ltrans-until-11I*[**where** *?n*=*TT* **and** *?G*=*G'*])
show *1* < *length* (*tpsC1* (*Suc* (*fmt n*)))
using *tpsC1-def* **by** *simp*
show ∀ *h*<*G'*. *ec1 h* < *G'*
using *ec1* **by** *simp*
show *lneigh* (*tps10* ! *1*) *StartSym TT*
using *begin-tape-def begin-tape-zip-cont tpsC1-def tpsL-def* **by** (*intro lneighI*) *simp-all*
show *Suc* (*fmt n*) $\leq$ *tpsC1* (*Suc* (*fmt n*)) :#: *1*
using *tpsC1-def tpsL-def* **by** *simp*
show *map* (*Pair* (*n* + *1*)) (*rev* [*0*..<*TT*]) @ [(*n* + *1*, *0*)] =
map (λ*i*. (*tps10* :#: *0*, *tps10* :#: *1* − *Suc i*)) [*0*..<*TT*] @ [(*tps10* :#: *0*, *tps10* :#: *1* − *TT*)]
proof −
have *1*: *tps10* :#: *0* = *n* + *1*
using *tpsC1-def tpsL-pos-0* **by** *simp*
moreover have *2*: *tps10* :#: *1* = *TT*
using *tpsC1-def tpsL-pos-1* **by** *simp*
ultimately have *map* (λ*i*. (*tps10* :#: *0*, *tps10* :#: *1* − *Suc i*)) [*0*..<*TT*] = *map* (λ*i*. (*n* + *1*, *TT* − *Suc i*)) [*0*..<*TT*]
by *simp*
moreover have *map* (λ*i*. (*c1*, *c2* − *Suc i*)) [*0*..<*c2*] = *map* (*Pair c1*) (*rev* [*0*..<*c2*]) **for** *c1 c2* :: *nat*
by (*intro nth-equalityI*, *simp*)

(*metis* (*no-types*, *lifting*) *add-cancel-left-left add-lessD1 diff-less diff-zero length-map length-upt nth-map-upt rev-map rev-nth zero-less-Suc*)
ultimately have *map* (*λi.* (*tps10* :#: *0*, *tps10* :#: *1* − *Suc i*)) [*0*..<*TT*] = *map* (*Pair* (*n* + *1*)) (*rev* [*0*..<*TT*])
by *metis*
then show *?thesis*
using *1 2* **by** *simp*
qed
show *tps11* = (*tpsC1 TT*) [*1* := *ltransplant* (*tpsC1 TT* ! *1*) (*tpsC1 TT* ! *1*) *ec1 TT*]
using *tps11-def* **by** *simp*
qed
qed

definition *es12* ≡ *es11* @ [(*n* + *1*, *1*)]

The upper bound on the length of the trace will help us establish an upper bound of the total running time.

lemma *length-es12*: *length es12* ≤ *length* (*es-fmt n*) + *43* ∗ *k* ∗ *TT* ∗ *TT*
proof −
have *length es12* ≤ *length* (*es-fmt n*) + *40* ∗ *k* ∗ *TT* ∗ *TT* + *3* ∗ *TT* ∗ *TT*
using *es12-def len-es11* **by** *simp*
moreover have *3* ∗ *TT* ∗ *TT* ≤ *3* ∗ *k* ∗ *TT* ∗ *TT*
proof −
have *3* ≤ *3* ∗ *k*
using *k-ge-2* **by** *simp*
then show *?thesis*
by (*meson mult-le-mono1*)
qed
ultimately show *?thesis*
by *linarith*
qed

definition *tps12* ≡ *tps11*[*1* := *tps11* ! *1* |:=| (*ec1* (*tps11* :.: *1*)) |+| *1*]

lemma *tm12*: *traces tm12 tps0 es12 tps12*
unfolding *tm12-def es12-def*
proof (*rule traces-sequential*[*OF tm11*])
show *traces* (*tm-rtrans 1 ec1*) *tps11* [(*n* + *1*, *1*)] *tps12*
proof (*rule traces-tm-rtrans-1I*)
show *1* < *length tps11*
using *tps11-def tpsC1-def* **by** *simp*
show [(*n* + *1*, *1*)] = [(*tps11* :#: *0*, *tps11* :#: *1* + *1*)]
using *tps11-def tpsC1-def tpsL-pos-0 tpsL-pos-1 ltransplant-def* **by** *simp*
show *tps12* = *tps11*[*1* := *tps11* ! *1* |:=| *ec1* (*tps11* :.: *1*) |+| *1*]
using *tps12-def* **by** *simp*
qed
qed

lemma *tps11-0*: (*tps11* ::: *1*) *0* = (*zip-cont TT* (*replicate k* (*0*, *Some 0*))) *0*
using *tps11-def tpsC1-def tpsL-def ltransplant-def* **by** *simp*

lemma *tps11-gr0-exec*:
assumes *i* > *0*
shows (*tps11* ::: *1*) *i* = (*exec TT* <:> *1*) *i*
proof −
let *?tp* = *tps10* ! *1*
let *?xs* = *replicate k* (*0*, *Some 0*)
have *1*: *tps11* ! *1* = *ltransplant ?tp ?tp ec1 TT*
using *tps11-def tpsC1-def tpsL-def* **by** *simp*
have *2*: *tps10* :#: *1* = *TT*
using *tpsC1-def tpsL-def* **by** *simp*
show *?thesis*
proof (*cases i* ≤ *TT*)

```
    case le-TT: True
    then have 0 < i ∧ i ≤ TT
      using assms by simp
    then have *: (tps11 ::: 1) i = ec1 (fst ?tp i)
      using 1 tpsC1-def tpsL-def ltransplant-def by simp
    show ?thesis
    proof (cases i = TT)
      case True
      then have ¬ is-code ((zip-cont TT ?xs) i)
        by (simp add: zip-cont-eq-0)
      then have (tps11 ::: 1) i = 0
        using * 2 True tpsC1-at-T by simp
      moreover have (exec TT <:> 1) TT = 0
        using tps-ge-TT-0 by simp
      ultimately show ?thesis
        using True by simp
    next
      case False
      then have i < TT
        using le-TT by simp
      then have fst ?tp i = (zip-cont TT ?xs) i
        using tpsC1-def tpsL-def by simp
      then have (tps11 ::: 1) i = ec1 ((zip-cont TT ?xs) i)
        using * by simp
      moreover have is-code ((zip-cont TT ?xs) i)
        using zip-cont-gt-1 zip-cont-less ‹i < TT› by simp
      ultimately have (tps11 ::: 1) i = enc-nth ((zip-cont TT ?xs) i) 1
        by simp
      then have (tps11 ::: 1) i = (exec TT <:> 1) i
        using enc-nth-def dec-zip-cont-less-k[OF ‹i < TT›] k-ge-2 by simp
      then show ?thesis
        by simp
    qed
  next
    case False
    then have (tps11 ::: 1) i = 0
      using 1 tpsC1-def tpsL-def ltransplant-def zip-cont-eq-0 by force
    moreover have (exec TT <:> 1) i = 0
      using False tps-ge-TT-0 by simp
    ultimately show ?thesis
      by simp
  qed
qed

definition tps12' ≡
    [(⌊zs⌋, n + 1),
     (exec TT <:> 1, 1),
     ⌈fst (exec TT)⌉] @
    map (λi. ⌈□⌉) [0..<k] @
    map (λi. ⌈□⌉) [0..<k]

lemma tps12': tps12' = tps12
proof (rule nth-equalityI)
  show length tps12' = length tps12
    using tps12'-def tps12-def tps11-def tpsC1-def by simp
  show tps12' ! j = tps12 ! j if j < length tps12' for j
  proof −
    have length tps12' = 2 * k + 3
      using tps12'-def by simp
    then consider j = 0 | j = 1 | j = 2 | 3 ≤ j ∧ j < k + 3 | k + 3 ≤ j ∧ j < 2 * k + 3
      using ‹j < length tps12'› by linarith
    then show ?thesis
    proof (cases)
```

```
  case 1
  then show ?thesis
    unfolding tps12'-def tps12-def tps11-def tpsC1-def tpsL-def by simp
next
  case 2
  then have lhs: tps12' ! j = (λi. (exec TT <:> 1) i, 1)
    using tps12'-def by simp
  let ?tp = tps10 ! 1
  let ?xs = replicate k (0, Some 0)
  have tps11 :#: 1 = 0
    using tps11-def ltransplant-def tpsC1-def tpsL-pos-1 by simp
  have rhs: tps12 ! j = (ltransplant ?tp ?tp ec1 TT) |:=| (ec1 (tps11 :.: 1)) |+| 1
    using tps12-def 2 ‹length tps12' = length tps12› tps11-def that by simp
  have tps10: tps10 ! j = (zip-cont TT ?xs, TT)
    using tpsC1-def 2 tpsL-1 by simp
  show tps12' ! j = tps12 ! j
  proof
    show tps12' :#: j = tps12 :#: j
      using lhs rhs ltransplant-def tps10 2 by simp
    have tps12: tps12 ! 1 = tps11 ! 1 |:=| (ec1 (tps11 :.: 1)) |+| 1
      using tps12-def 2 ‹length tps12' = length tps12› that by auto
    have (tps12' ::: 1) i = (tps12 ::: 1) i for i
    proof (cases i = 0)
      case True
      then have (tps12 ::: 1) i = ec1 (tps11 :.: 1)
        using tps12 ‹tps11 :#: 1 = 0› by simp
      moreover have (tps11 :.: 1) = (zip-cont TT ?xs) 0
        using tps11-0 ‹tps11 :#: 1 = 0› by simp
      ultimately have (tps12 ::: 1) i = ec1 ((zip-cont TT ?xs) 0)
        by simp
      moreover have is-code ((zip-cont TT ?xs) 0)
        using zip-cont-gt-1 zip-cont-less by simp
      ultimately have (tps12 ::: 1) i = enc-nth ((zip-cont TT ?xs) 0) 1
        by simp
      then have (tps12 ::: 1) i = enc-nth (zip-cont TT ?xs i) 1
        using True by simp
      then have (tps12 ::: 1) i = (exec TT <:> 1) i
        using enc-nth-def dec-zip-cont-less-k True k-ge-2 by simp
      then show ?thesis
        using tps12'-def by simp
    next
      case False
      then have (tps12 ::: 1) i = (tps11 ::: 1) i
        using tps12 ‹tps11 :#: 1 = 0› by simp
      then have (tps12 ::: 1) i = (exec TT <:> 1) i
        using False tps11-gr0-exec by simp
      moreover have (tps12' ::: 1) i = (exec TT <:> 1) i
        using tps12'-def by simp
      ultimately show ?thesis
        by simp
    qed
    then show tps12' ::: j = tps12 ::: j
      using 2 by auto
  qed
next
  case 3
  then show ?thesis
    unfolding tps12'-def tps12-def tps11-def tpsC1-def tpsL-def by simp
next
  case 4
  then show ?thesis
    unfolding tps12'-def tps12-def tps11-def tpsC1-def
    using tpsL-mvs' threeplus2k-2[where ?a=(⌊zs⌋, n + 1)]
```

by *simp*
next
case *5*
then show *?thesis*
unfolding *tps12′-def tps12-def tps11-def tpsC1-def*
using *tpsL-symbs′ threeplus2k-3*[**where** *?a*=(⌊*zs*⌋, *n* + *1*)]
by *simp*
qed
qed
qed

lemma *tm12′*: *traces tm12 tps0 es12 tps12′*
using *tm12 tps12′* **by** *simp*

end

5.3.4 Shrinking the Turing machine to two tapes

The simulator TM *tm12* has $2k+3$ tapes, of which $2k+1$ are immobile and thus can be removed by the memorization-in-states technique, resulting in a two-tape TM.

lemma *immobile-tm12*:
assumes *j* > *1* **and** *j* < *2* ∗ *k* + *3*
shows *immobile tm12 j* (*2* ∗ *k* + *3*)
proof −
have *immobile tm1 j* (*2* ∗ *k* + *3*)
unfolding *tm1-def*
using *immobile-append-tapes*[*of j 2*∗*k*+*3*, *OF - - - fmt*(*1*)] *assms*
by *simp*
moreover have *immobile tm1-2 j* (*2* ∗ *k* + *3*)
using *tm1-2-def tm-const-until-def immobile-tm-trans-until assms* **by** *simp*
ultimately have *immobile tm2 j* (*2* ∗ *k* + *3*)
using *tm2-def immobile-sequential tm1-2-tm tm1-tm* **by** *simp*
then have *immobile tm3 j* (*2* ∗ *k* + *3*)
using *tm3-def immobile-sequential*[*OF tm2-tm*] *tm-start-tm immobile-tm-start assms G′-ge* **by** *simp*
then have *immobile tm4 j* (*2* ∗ *k* + *3*)
using *tm4-def immobile-sequential*[*OF tm3-tm tm3-4-tm*] *immobile-tm-write assms* **by** *simp*
then have *immobile tm5 j* (*2* ∗ *k* + *3*)
using *tm5-def immobile-sequential*[*OF tm4-tm*] *G′-ge*(*1*) *immobile-tm-right tm-right-tm assms* **by** *simp*
then have *immobile tm6 j* (*2* ∗ *k* + *3*)
using *tm6-def immobile-sequential*[*OF tm5-tm tm5-6-tm*] *immobile-tm-trans-until tm5-6-def assms* **by** *simp*
then have *immobile tm7 j* (*2* ∗ *k* + *3*)
using *tm7-def immobile-sequential*[*OF tm6-tm tm-left-until1-tm*] *immobile-tm-left-until assms* **by** *simp*
then have *immobile tm8 j* (*2* ∗ *k* + *3*)
using *tm8-def immobile-sequential*[*OF tm7-tm*] *immobile-tm-write assms G′-ge tm-write-tm* **by** *simp*
then have *9*: *immobile tm9 j* (*2* ∗ *k* + *3*)
using *tm9-def immobile-sequential*[*OF tm8-tm*] *immobile-tm-write-many tm-write-many-tm k-ge-2 G′-ge assms*
by *simp*

have *C*: *immobile tmC j* (*2* ∗ *k* + *3*)
unfolding *tmC-def tm-right-until-def tm-cp-until-def*
using *tm-cp-until-tm immobile-tm-trans-until G′-ge*(*1*) *assms*
by *simp*

have *cmdL2 rs* [~] *j* = *Stay* **if** *length rs* = *2* ∗ *k* + *3* **for** *rs*
proof (*cases rs* ! *1* = □)
case *True*
then show *?thesis*
using *cmdL2-def assms that* **by** *simp*
next
case *False*
then consider *j* = *2* | *3* ≤ *j* ∧ *j* < *3* + *k* | *3* + *k* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
using *assms* **by** *linarith*
then show *?thesis*

```
  proof (cases)
    case 1
    then show ?thesis
      using cmdL2-def assms that by simp
  next
    case 2
    then show ?thesis
      using assms that cmdL2-at-3 False by simp
  next
    case 3
    then show ?thesis
      using assms that cmdL2-at-4 False by simp
  qed
qed
then have immobile tmL1-2 j (2 * k + 3)
  using tmL1-2-def by auto
then have immobile tmL2 j (2 * k + 3)
  unfolding tmL2-def tmL1-def
  using tm-left-until1-tm immobile-tm-left-until tmL2-tm immobile-sequential tmL1-2-tm assms
  by auto
moreover have cmdL3 rs [~] j = Stay if length rs = 2 * k + 3 for rs
proof -
  consider j = 2 | 3 ≤ j ∧ j < 3 + k | 3 + k ≤ j ∧ j < 2 * k + 3
    using assms by linarith
  then show ?thesis
  proof (cases)
    case 1
    then show ?thesis
      using cmdL3-def assms that by simp
  next
    case 2
    then show ?thesis
      using assms that cmdL3-at-3a cmdL3-at-3b
      by (metis (no-types, lifting) add.commute prod.sel(2))
  next
    case 3
    then show ?thesis
      using assms that cmdL3-at-4a cmdL3-at-4b
      by (metis (no-types, lifting) add.commute prod.sel(2))
  qed
qed
ultimately have immobile tmL3 j (2 * k + 3)
  unfolding tmL3-def
  using tmL2-tm immobile-sequential assms tmL2-3-def tmL2-3-tm immobile-def
  by simp
then have L4: immobile tmL4 j (2 * k + 3)
  unfolding tmL4-def
  using tmL3-tm immobile-sequential assms tm-left-until1-tm immobile-tm-left-until
  by auto

have (cmdL5 jj) rs [~] j = Stay if length rs = 2 * k + 3 and jj < k for rs jj
proof (cases rs ! 1 = □)
  case True
  then show ?thesis
    using cmdL5-eq-0 assms that by simp
next
  case False
  then consider j = 2 | 3 ≤ j ∧ j < 3 + k | 3 + k ≤ j ∧ j < 2 * k + 3
    using assms by linarith
  then show ?thesis
  proof (cases)
    case 1
    then show ?thesis
```

using *that* **by** (*simp add*: *cmdL5-def*)
next
case *2*
then show *?thesis*
using *assms that cmdL5-at-3 False* **by** *simp*
next
case *3*
then show *?thesis*
using *assms that cmdL5-at-4 False* **by** *simp*
qed
qed
then have *immobile* (*tmL45 jj*) *j* (*2* ∗ *k* + *3*) **if** *jj* < *k* **for** *jj*
by (*auto simp add*: *that tmL45-def*)
then have *L46*: *immobile* (*tmL46 jj*) *j* (*2* ∗ *k* + *3*) **if** *jj* < *k* **for** *jj*
using *tmL46-def immobile-sequential*[*OF tmL45-tm*] *tm-left-tm immobile-tm-left assms that k-ge-2 G′-ge* **by** *simp*

have (*cmdL7 jj*) *rs* [~] *j* = *Stay* **if** *length rs* = *2* ∗ *k* + *3* **and** *jj* < *k* **for** *rs jj*
proof −
consider (*a*) *condition7a rs jj* | (*b*) *condition7b rs jj* | (*c*) *condition7c rs jj*
by *blast*
then show *?thesis*
proof (*cases*)
case *a*
consider *j* = *2* | *3* ≤ *j* ∧ *j* < *k* + *3* | *3* + *k* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
using *assms* **by** *linarith*
then show *?thesis*
using *cmdL7-def a threeplus2k-2*[*of - -* (*rs* ! *0*, *Stay*)] *threeplus2k-3*[*of - -* (*rs* ! *0*, *Stay*)]
by (*cases*) *simp-all*
next
case *b*
consider *j* = *2* | *3* ≤ *j* ∧ *j* < *k* + *3* | *3* + *k* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
using *assms* **by** *linarith*
then show *?thesis*
using *cmdL7-def b threeplus2k-2*[*of - -* (*rs* ! *0*, *Stay*)] *threeplus2k-3*[*of - -* (*rs* ! *0*, *Stay*)]
by (*cases*) *simp-all*
next
case *c*
consider *j* = *2* | *3* ≤ *j* ∧ *j* < *k* + *3* | *3* + *k* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
using *assms* **by** *linarith*
then show *?thesis*
using *cmdL7-def c threeplus2k-2*[*of - -* (*rs* ! *0*, *Stay*)] *threeplus2k-3*[*of - -* (*rs* ! *0*, *Stay*)]
by (*cases*) *simp-all*
qed
qed
then have *immobile* (*tmL67 jj*) *j* (*2* ∗ *k* + *3*) **if** *jj* < *k* **for** *jj*
by (*auto simp add*: *that tmL67-def*)
then have *L47*: *immobile* (*tmL47 jj*) *j* (*2* ∗ *k* + *3*) **if** *jj* < *k* **for** *jj*
using *tmL47-def immobile-sequential*[*OF tmL46-tm tmL67-tm*] *L46 assms that* **by** *simp*

have (*cmdL8 jj*) *rs* [~] *j* = *Stay* **if** *length rs* = *2* ∗ *k* + *3* **and** *jj* < *k* **for** *rs jj*
proof −
consider (*a*) *condition8a rs jj* | (*b*) *condition8b rs jj* | (*c*) *condition8c rs jj* | (*d*) *condition8d rs jj*
by *blast*
then show *?thesis*
proof (*cases*)
case *a*
consider *j* = *2* | *3* ≤ *j* ∧ *j* < *k* + *3* | *3* + *k* ≤ *j* ∧ *j* < *2* ∗ *k* + *3*
using *assms* **by** *linarith*
then show *?thesis*
using *cmdL8-def a threeplus2k-2*[*of - -* (*rs* ! *0*, *Stay*)] *threeplus2k-3*[*of - -* (*rs* ! *0*, *Stay*)]
by (*cases*) *simp-all*
next

case *b*
consider $j = 2 \mid 3 \leq j \wedge j < k + 3 \mid 3 + k \leq j \wedge j < 2 * k + 3$
using *assms* **by** *linarith*
then show *?thesis*
using *cmdL8-def b threeplus2k-2*[*of - -* (*rs* ! *0*, *Stay*)] *threeplus2k-3*[*of - -* (*rs* ! *0*, *Stay*)]
by (*cases*) *simp-all*
next
case *c*
consider $j = 2 \mid 3 \leq j \wedge j < k + 3 \mid 3 + k \leq j \wedge j < 2 * k + 3$
using *assms* **by** *linarith*
then show *?thesis*
using *cmdL8-def c threeplus2k-2*[*of - -* (*rs* ! *0*, *Stay*)] *threeplus2k-3*[*of - -* (*rs* ! *0*, *Stay*)]
by (*cases*) *simp-all*
next
case *d*
consider $j = 2 \mid 3 \leq j \wedge j < k + 3 \mid 3 + k \leq j \wedge j < 2 * k + 3$
using *assms* **by** *linarith*
then show *?thesis*
using *cmdL8-def d threeplus2k-2*[*of - -* (*rs* ! *0*, *Stay*)] *threeplus2k-3*[*of - -* (*rs* ! *0*, *Stay*)]
by (*cases*) *simp-all*
qed
qed
then have *immobile* (*tmL78 jj*) *j* $(2 * k + 3)$ **if** $jj < k$ **for** *jj*
by (*auto simp add*: *that tmL78-def*)
then have *immobile* (*tmL48 jj*) *j* $(2 * k + 3)$ **if** $jj < k$ **for** *jj*
using *tmL48-def immobile-sequential*[*OF tmL47-tm tmL78-tm*] *L47 assms that* **by** *simp*
then have *L49*: *immobile* (*tmL49 jj*) *j* $(2 * k + 3)$ **if** $jj < k$ **for** *jj*
using *tmL49-def immobile-sequential*[*OF tmL48-tm*] *tm-left-until1-tm immobile-tm-left-until assms that* **by** *simp*

have *L49-upt*: *immobile* (*tmL49-upt* j') *j* $(2 * k + 3)$ **if** $j' \leq k$ **for** j'
using *that*
proof (*induction* j')
case *0*
then show *?case*
by *auto*
next
case (*Suc* j')
have *tmL49-upt* (*Suc* j') = *tmL49-upt* j' ;; *tmL49* j'
by *simp*
moreover have *turing-machine* $(2*k+3)$ G' (*tmL49-upt* j')
using *tmL49-upt-tm Suc* **by** *simp*
moreover have *immobile* (*tmL49-upt* j') *j* $(2*k+3)$
using *Suc* **by** *simp*
moreover have *turing-machine* $(2*k+3)$ G' (*tmL49* j')
using *tmL49-tm Suc* **by** *simp*
moreover have *immobile* (*tmL49* j') *j* $(2*k+3)$
using *L49 Suc* **by** *simp*
ultimately show *?case*
using *immobile-sequential* **by** *simp*
qed
then have *immobile tmL9 j* $(2 * k + 3)$
using *tmL9-def immobile-sequential*[*OF tmL4-tm tmL49-upt-tm*] *L4* **by** *simp*
then have *L10*: *immobile tmL10 j* $(2 * k + 3)$
using *tmL10-def immobile-sequential*[*OF tmL9-tm tmC-tm*] *C* **by** *simp*

have *cmdL11 rs* [~] *j* = *Stay* **if** *length rs* $= 2 * k + 3$ **and** $jj < k$ **for** *rs jj*
proof −
consider $j = 2 \mid 3 \leq j \wedge j < 3 + k \mid 3 + k \leq j \wedge j < 2 * k + 3$
using *assms* **by** *linarith*
then show *?thesis*
proof (*cases*)
case *1*

then show *?thesis*
by (*simp add*: *cmdL11-def*)
next
case *2*
then show *?thesis*
using *cmdL11-def threeplus2k-2*[**where** *?a*=(*rs* ! *0*, *Stay*)] **by** *simp*
next
case *3*
then show *?thesis*
using *cmdL11-def threeplus2k-3*[**where** *?a*=(*rs* ! *0*, *Stay*)] **by** *simp*
qed
qed
then have *immobile* [*cmdL11*] *j* (*2* ∗ *k* + *3*)
using *k-ge-2 assms* **by** *force*
then have *immobile tmL11 j* (*2* ∗ *k* + *3*)
using *tmL11-def immobile-sequential*[*OF tmL10-tm tmL1011-tm*] *L10* **by** *simp*
then have *immobile tmL12 j* (*2* ∗ *k* + *3*)
using *tmL12-def immobile-sequential*[*OF tmL11-tm tm-left-until1-tm*] *immobile-tm-left-until assms* **by** *simp*
then have *immobile tmL13 j* (*2* ∗ *k* + *3*)
using *tmL13-def immobile-sequential*[*OF tmL12-tm tm-write-many-tm*] *immobile-tm-write-many*
assms k-ge-2 G′-ge(*1*)
by *simp*
then have *immobile tmLoop j* (*2* ∗ *k* + *3*)
using *tmLoop-def C immobile-loop*[*OF tmC-tm tmL13-tm*] *assms*(*2*) **by** *simp*
then have *immobile tm10 j* (*2* ∗ *k* + *3*)
using *tm10-def immobile-sequential*[*OF tm9-tm tmLoop-tm*] *9* **by** *simp*
then have *immobile tm11 j* (*2* ∗ *k* + *3*)
using *tm11-def immobile-sequential*[*OF tm10-tm tm-ltrans-until-tm*] *ec1 G′-ge immobile-tm-ltrans-until assms*
by *simp*
then show *immobile tm12 j* (*2* ∗ *k* + *3*)
using *tm12-def immobile-sequential*[*OF tm11-tm tm-rtrans-tm*] *ec1 G′-ge immobile-tm-rtrans assms* **by** *simp*
qed

definition *tps12″ zs* ≡
[(⌊*zs*⌋, *length zs* + *1*),
(*exec zs* (*Suc* (*fmt* (*length zs*))) <:> *1*, *1*)]

lemma *tps12″*:
assumes *bit-symbols zs*
shows *tps12″ zs* = *take 2* (*tps12′ zs*)
using *tps12′-def tps12″-def assms* **by** *simp*

This is the actual simulator Turing machine we are constructing in this section. It is *tm12* stripped of all memorization tapes:

definition *tmO2T* ≡ *icartesian* (*2* ∗ *k* + *1*) *tm12 G′*

lemma *tmO2T-tm*: *turing-machine 2 G′ tmO2T*
unfolding *tmO2T-def*
using *immobile-tm12 tm12-tm icartesian-tm*[*of 2* ∗ *k* + *1 G′*]
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-le-lessD add.assoc add-less-mono1 le-add2*
numeral-3-eq-3 one-add-one plus-1-eq-Suc)

The constructed two-tape Turing machine computes the same output as the original Turing machine.

lemma *tmO2T*:
assumes *bit-symbols zs*
shows *traces tmO2T* (*snd* (*start-config 2 zs*)) (*es12 zs*) (*tps12″ zs*)
proof −
have ∗: *2* ∗ *k* + *1* + *2* = *2* ∗ *k* + *3*
by *simp*
then have *turing-machine* (*2* ∗ *k* + *1* + *2*) *G′ tm12*
using *tm12-tm* **by** *metis*
moreover have ⋀*j*. *j* < *2* ∗ *k* + *1* ⟹ *immobile tm12* (*j* + *2*) (*2* ∗ *k* + *1* + *2*)
using *immobile-tm12 immobile-def* **by** *simp*

moreover have $\forall$ *i<length zs. zs ! i < G′*
  using *assms G′-ge-G zs-less-G* **by** (*meson order-less-le-trans*)
moreover have *traces tm12* (*snd* (*start-config* (*2* ∗ *k* + *1* + *2*) *zs*)) (*es12 zs*) (*tps12′ zs*)
  using *tm12′ tps0-start-config assms* ∗ **by** (*metis* (*no-types*, *lifting*) *prod.sel*(*2*))
ultimately show *?thesis*
  using *icartesian*[*of 2* ∗ *k* + *1 G′ tm12 zs es12 zs tps12′ zs*] *tmO2T-def tps12′′ assms* **by** *simp*
qed

5.3.5 Time complexity

This is the bound for the running time of *tmO2T*:

definition *TTT* :: *nat* ⇒ *nat* **where**
  TTT ≡ λ*n. length* (*es-fmt n*) + *43* ∗ *k* ∗ *Suc* (*fmt n*) ∗ *Suc* (*fmt n*)

lemma *execute-tmO2T*:
  assumes *bit-symbols zs*
  shows *execute tmO2T* (*start-config 2 zs*) (*TTT* (*length zs*)) = (*length tmO2T*, *tps12′′ zs*)
proof −
  have *trace tmO2T* (*start-config 2 zs*) (*es12 zs*) (*length tmO2T*, *tps12′′ zs*)
    using *tmO2T assms traces-def start-config-def* **by** *simp*
  then have *execute tmO2T* (*start-config 2 zs*) (*length* (*es12 zs*)) = (*length tmO2T*, *tps12′′ zs*)
    using *trace-def* **by** *simp*
  moreover have *length* (*es12 zs*) ≤ *TTT* (*length zs*)
    using *assms length-es12 TTT-def* **by** *simp*
  ultimately show *?thesis*
    by (*metis execute-after-halting-ge fst-conv*)
qed

The simulator TM *tmO2T* halts with the output tape head on cell 1.

lemma *execute-tmO2T-1*:
  assumes *bit-symbols zs*
  shows *execute tmO2T* (*start-config 2 zs*) (*TTT* (*length zs*)) <!> *1* =
    (*execute M* (*start-config k zs*) (*T* (*length zs*)) <:> *1*, *1*)
proof −
  have *T* (*length zs*) ≤ *Suc* (*fmt* (*length zs*))
    by (*metis add-leD1 le-Suc-eq fmt*(*4*) *T′-def*)
  then have ∗: *execute M* (*start-config k zs*) (*T* (*length zs*)) =
      execute M (*start-config k zs*) (*Suc* (*fmt* (*length zs*)))
    using *execute-after-halting-ge time-bound-T time-bound-def assms* **by** (*metis* (*no-types*, *lifting*))

  have *execute tmO2T* (*start-config 2 zs*) (*TTT* (*length zs*)) = (*length tmO2T*, *tps12′′ zs*)
    using *assms execute-tmO2T* **by** *simp*
  then have *execute tmO2T* (*start-config 2 zs*) (*TTT* (*length zs*)) <!> *1* =
      (*execute M* (*start-config k zs*) (*Suc* (*fmt* (*length zs*))) <:> *1*, *1*)
    using *tps12′′-def exec-def assms* **by** *simp*
  then show *?thesis*
    using ∗ **by** *simp*
qed

lemma *poly-TTT*: *big-oh-poly TTT*
proof −
  have *1*: *big-oh-poly* (λ*n. length* (*es-fmt n*))
    using *fmt*(*2*) **by** *simp*
  have *big-oh-poly* (λ*n. fmt n* + *1*)
    using *fmt*(*3*) *big-oh-poly-const big-oh-poly-sum* **by** *blast*
  then have *big-oh-poly* (λ*n. Suc* (*fmt n*))
    by *simp*
  then have *big-oh-poly* (λ*n. Suc* (*fmt n*) ∗ *Suc* (*fmt n*))
    using *big-oh-poly-prod* **by** *blast*
  moreover have *big-oh-poly* (λ*n. 43* ∗ *k*)
    using *big-oh-poly-const* **by** *simp*
  ultimately have *big-oh-poly* (λ*n. 43* ∗ *k* ∗ (*Suc* (*fmt n*) ∗ *Suc* (*fmt n*)))
    using *big-oh-poly-prod* **by** *blast*

moreover have ($\lambda n.$ *43* $*$ *k* $*$ (*Suc* (*fmt n*) $*$ *Suc* (*fmt n*))) = ($\lambda n.$ *43* $*$ *k* $*$ *Suc* (*fmt n*) $*$ *Suc* (*fmt n*))
  by (*metis* (*mono-tags*, *opaque-lifting*) *mult.assoc*)
ultimately have *big-oh-poly* ($\lambda n.$ *43* $*$ *k* $*$ *Suc* (*fmt n*) $*$ *Suc* (*fmt n*))
  by *simp*
then have *big-oh-poly* ($\lambda n.$ *length* (*es-fmt n*) + *43* $*$ *k* $*$ *Suc* (*fmt n*) $*$ *Suc* (*fmt n*))
  using *1 big-oh-poly-sum* **by** *simp*
then show *?thesis*
  unfolding *TTT-def* **by** *simp*
qed

5.3.6 Obliviousness

The two-tape simulator machine is oblivious.

lemma *tmO2T-oblivious*:
  assumes *length zs1* = *length zs2* **and** *bit-symbols zs1* **and** *bit-symbols zs2*
  shows *es12 zs1* = *es12 zs2*
proof –
  have *es1 zs1* = *es1 zs2*
    using *es1-def assms* **by** *simp*

  moreover have *es1-2 zs1* = *es1-2 zs2*
    using *es1-2-def assms* **by** (*metis* (*no-types*, *lifting*))
  ultimately have *es2 zs1* = *es2 zs2*
    using *es2-def assms* **by** *simp*
  then have *es3 zs1* = *es3 zs2*
    using *es3-def assms* **by** *simp*
  then have *es4 zs1* = *es4 zs2*
    using *es4-def assms* **by** *simp*
  then have *es5 zs1* = *es5 zs2*
    using *es5-def assms* **by** *simp*
  then have *es6 zs1* = *es6 zs2*
    using *es6-def assms* **by** *simp*
  then have *es7 zs1* = *es7 zs2*
    using *es7-def assms* **by** *simp*
  then have *es8 zs1* = *es8 zs2*
    using *es8-def assms* **by** *simp*
  then have *9*: *es9 zs1* = *es9 zs2*
    using *es9-def assms* **by** *simp*

  have *C*: *esC zs1 t* = *esC zs2 t* **for** *t*
    using *esC-def assms* **by** *simp*
  then have *Loop-break*: *esLoop-break zs1* = *esLoop-break zs2*
    using *esLoop-break-def tpsC1-def tpsL-def assms* **by** *simp*

  have *esL1 zs1* = *esL1 zs2*
    using *esL1-def assms* **by** *auto*
  moreover have *esL1-2 zs1* = *esL1-2 zs2*
    using *esL1-2-def assms* **by** *simp*
  ultimately have *esL2 zs1* = *esL2 zs2*
    using *esL2-def assms* **by** *auto*
  then have *esL3 zs1* = *esL3 zs2*
    using *esL3-def assms* **by** *auto*
  then have *L4*: *esL4 zs1* = *esL4 zs2*
    using *esL4-def assms* **by** *auto*

  have *esL45 zs1* = *esL45 zs2*
    using *esL45-def assms* **by** *simp*
  then have *esL46 zs1* = *esL46 zs2*
    using *esL46-def assms* **by** *simp*
  moreover have *esL67 zs1* = *esL67 zs2*
    using *esL67-def assms* **by** *simp*
  ultimately have *esL47 zs1* = *esL47 zs2*
    using *esL47-def assms* **by** *simp*

moreover have *esL78 zs1 = esL78 zs2*
using *esL78-def assms* **by** *simp*
ultimately have *esL48 zs1 = esL48 zs2*
using *esL48-def assms* **by** *simp*
then have *esL49 zs1 = esL49 zs2*
using *esL49-def assms* **by** *simp*
then have *esL49-upt zs1 = esL49-upt zs2*
using *esL49-upt-def assms* **by** *simp*
then have *esL9 zs1 = esL9 zs2*
using *esL9-def L4 assms* **by** *auto*
then have *esL10 zs1 = esL10 zs2*
using *esL10-def C assms* **by** *auto*
then have *esL11 zs1 = esL11 zs2*
using *esL11-def assms* **by** *auto*
then have *esL12 zs1 = esL12 zs2*
using *esL12-def assms* **by** *auto*
then have *esL13 zs1 = esL13 zs2*
using *esL13-def assms* **by** *auto*
then have *esLoop-while zs1 = esLoop-while zs2*
using *esLoop-while-def C tpsC1-def tpsL13-def tpsL-def assms* **by** *auto*
then have *esLoop zs1 = esLoop zs2*
using *esLoop-def Loop-break assms* **by** *auto*
then have *es10 zs1 = es10 zs2*
using *es10-def 9 assms* **by** *auto*
then have *es11 zs1 = es11 zs2*
using *es11-def assms* **by** *simp*
then show *es12 zs1 = es12 zs2*
using *es12-def assms* **by** *simp*
qed

end

5.4 $\mathcal{NP}$ and obliviousness

This section presents the main result of this chapter: For every language $L \in \mathcal{NP}$ there is a polynomial-time two-tape oblivious verifier TM that halts with the output tape head on a **1** symbol iff. in the input $\langle x, u\rangle$, the string u is a certificate for x. The proof combines two lemmas. First *NP-output-len-1*, which says that we can assume the verifier outputs only one symbol (namely, **0** or **1**), and second *two-tape.execute-tmO2T-1*, which says that the two-tape oblivious TM halts with output tape head in cell 1. This cell will contain either **0** or **1** by the first lemma.

lemma *NP-imp-oblivious-2tape*:
fixes L :: *language*
assumes $L \in \mathcal{NP}$
obtains M G T p **where**
big-oh-poly T **and**
polynomial p **and**
turing-machine 2 G M **and**
oblivious M **and**
$\bigwedge y$. *bit-symbols y* $\Longrightarrow$ *fst* (*execute M* (*start-config 2 y*) (*T* (*length y*))) = *length M* **and**
$\bigwedge x$. $x \in L \longleftrightarrow$ ($\exists$ *u*. *length u* = *p* (*length x*) $\wedge$ *execute M* (*start-config 2* $\langle x;\ u\rangle$) (*T* (*length* $\langle x;\ u\rangle$)) <.> *1* = **1**)
proof −
define *Q* **where** *Q* = (λ*L k G M p T fverify*.
turing-machine k G M $\wedge$
polynomial p $\wedge$
big-oh-poly T $\wedge$
computes-in-time k M fverify T $\wedge$
($\forall$ *y*. *length* (*fverify y*) = *1*) $\wedge$
($\forall$ *x*. (*x* $\in$ *L*) = ($\exists$ *u*. *length u* = *p* (*length x*) $\wedge$ *fverify* $\langle x,\ u\rangle$ = [$\mathbb{I}$])))
have $\mathcal{NP}$ = {*L* :: *language*. $\exists$ *k G M p T fverify*. *Q L k G M p T fverify*}
unfolding *NP-output-len-1 Q-def* **by** *simp*
then obtain *k G M p T fverify* **where** *Q L k G M p T fverify*

```
  using assms by blast
then have alt:
  turing-machine k G M
  polynomial p
  big-oh-poly T
  computes-in-time k M fverify T
  ⋀y. length (fverify y) = 1
  ⋀x. (x ∈ L) = (∃ u. length u = p (length x) ∧ fverify ⟨x, u⟩ = [𝕀])
  using Q-def by simp-all

have tm-M: turing-machine k G M
  using alt(1) .
have poly-T: big-oh-poly T
  using alt(3) .
have time-bound-T: time-bound M k T
  unfolding time-bound-def
proof standard+
  fix zs
  assume zs: bit-symbols zs
  define x where x = symbols-to-string zs
  then have zs = string-to-symbols x
    using bit-symbols-to-symbols zs by simp
  then show fst (execute M (start-config k zs) (T (length zs))) = length M
    using computes-in-time-def alt(4)
    by (metis (no-types, lifting) execute-after-halting-ge length-map running-timeD(1))
qed

interpret two: two-tape M k G T
  using tm-M poly-T time-bound-T two-tape-def by simp

let ?M = two.tmO2T
let ?T = two.TTT
let ?G = two.G′
have big-oh-poly ?T
  using two.poly-TTT .
moreover have polynomial p
  using alt(2) .
moreover have turing-machine 2 ?G ?M
  using two.tmO2T-tm .
moreover have oblivious ?M
proof −
  let ?e = λn. two.es12 (replicate n 2)
  have ∃ tps. trace ?M (start-config 2 zs) (?e (length zs)) (length ?M, tps)
      if bit-symbols zs for zs
  proof −
    have traces ?M (snd (start-config 2 zs)) (two.es12 zs) (two.tps12′′ zs)
      using that two.tmO2T by simp
    then have ∗: trace ?M (start-config 2 zs) (two.es12 zs) (length ?M, two.tps12′′ zs)
      using start-config-def traces-def by simp

    let ?r = replicate (length zs) 2
    have length zs = length ?r
      by simp
    then have two.es12 zs = two.es12 ?r
      using two.tmO2T-oblivious that by (metis nth-replicate)
    then have trace ?M (start-config 2 zs) (?e (length zs)) (length ?M, two.tps12′′ zs)
      using ∗ by simp
    then show ?thesis
      by auto
  qed
  then show ?thesis
    using oblivious-def by fast
qed
```

moreover have *fst* (*execute ?M* (*start-config 2 y*) (*?T* (*length y*))) = *length ?M* **if** *bit-symbols y* **for** *y*
using *that two.execute-tmO2T* **by** *simp*
moreover have *x* ∈ *L* ⟷ (∃ *u. length u* = *p* (*length x*) ∧ *execute ?M* (*start-config 2* ⟨*x*; *u*⟩) (*?T* (*length* ⟨*x*; *u*⟩)) <.> *1* = **1**)
(**is** *?lhs* = *?rhs*) **for** *x*
proof
show *?lhs* ⟹ *?rhs*
proof −
assume *?lhs*
then have ∃ *u. length u* = *p* (*length x*) ∧ *fverify* ⟨*x*, *u*⟩ = [𝕀]
using *alt*(*6*) **by** *simp*
then obtain *u* **where** *u*: *length u* = *p* (*length x*) *fverify* ⟨*x*, *u*⟩ = [𝕀]
by *auto*

let *?y* = *fverify* ⟨*x*, *u*⟩
let *?cfg* = *execute M* (*start-config k* ⟨*x*; *u*⟩) (*T* (*length* ⟨*x*, *u*⟩))
have *computes-in-time k M fverify T*
using *alt*(*4*) **by** *simp*
then have *cfg*: *?cfg* <:> *1* = *string-to-contents ?y*
using *computes-in-time-execute* **by** *simp*

have *bit-symbols* ⟨*x*; *u*⟩
by *simp*
then have *execute ?M* (*start-config 2* ⟨*x*; *u*⟩) (*?T* (*length* ⟨*x*; *u*⟩)) <!> *1* =
(*execute M* (*start-config k* ⟨*x*; *u*⟩) (*T* (*length* ⟨*x*; *u*⟩)) <:> *1*, *1*)
using *two.execute-tmO2T-1* **by** *blast*
then have *execute ?M* (*start-config 2* ⟨*x*; *u*⟩) (*?T* (*length* ⟨*x*; *u*⟩)) <!> *1* =
(*string-to-contents ?y*, *1*)
using *cfg* **by** *simp*
then have *execute ?M* (*start-config 2* ⟨*x*; *u*⟩) (*?T* (*length* ⟨*x*; *u*⟩)) <!> *1* =
(*string-to-contents* [𝕀], *1*)
using *u*(*2*) **by** *auto*
moreover have |.| (*string-to-contents* [𝕀], *1*) = **1**
by *simp*
ultimately have *execute ?M* (*start-config 2* ⟨*x*; *u*⟩) (*?T* (*length* ⟨*x*; *u*⟩)) <.> *1* = **1**
by *simp*
then show *?thesis*
using *u*(*1*) **by** *auto*
qed
show *?rhs* ⟹ *?lhs*
proof −
assume *?rhs*
then obtain *u* **where** *u*:
length u = *p* (*length x*)
execute ?M (*start-config 2* ⟨*x*; *u*⟩) (*?T* (*length* ⟨*x*; *u*⟩)) <.> *1* = **1**
by *auto*
let *?zs* = ⟨*x*; *u*⟩
have *bit-symbols* ⟨*x*; *u*⟩
by *simp*
then have ∗: *execute ?M* (*start-config 2 ?zs*) (*?T* (*length ?zs*)) <!> *1* =
(*execute M* (*start-config k ?zs*) (*T* (*length ?zs*)) <:> *1*, *1*)
using *two.execute-tmO2T-1* **by** *blast*

let *?y* = *fverify* ⟨*x*, *u*⟩
let *?cfg* = *execute M* (*start-config k ?zs*) (*T* (*length* ⟨*x*, *u*⟩))
have *computes-in-time k M fverify T*
using *alt*(*4*) **by** *simp*
then have *cfg*: *?cfg* <:> *1* = *string-to-contents ?y*
using *computes-in-time-execute* **by** *simp*
then have *execute ?M* (*start-config 2 ?zs*) (*?T* (*length ?zs*)) <!> *1* =
(*string-to-contents* (*fverify* ⟨*x*, *u*⟩), *1*)
using ∗ **by** *simp*
then have *execute ?M* (*start-config 2 ?zs*) (*?T* (*length ?zs*)) <.> *1* =

string-to-contents (*fverify* ⟨*x*, *u*⟩) *1*
by *simp*
then have ∗∗: *string-to-contents* (*fverify* ⟨*x*, *u*⟩) *1* = **1**
using *u*(*2*) **by** *simp*

have *length* (*fverify* ⟨*x*, *u*⟩) = *1*
using *alt*(*5*) **by** *simp*
then have *string-to-contents* (*fverify* ⟨*x*, *u*⟩) *1* =
(*if fverify* ⟨*x*, *u*⟩ ! *0 then* **1** *else* **0**)
by *simp*
then have (*if fverify* ⟨*x*, *u*⟩ ! *0 then* **1** *else* **0**) = **1**
using ∗∗ **by** *auto*
then have *fverify* ⟨*x*, *u*⟩ ! *0* = 𝕀
by (*metis numeral-eq-iff semiring-norm*(*89*))
moreover have *y* = [𝕀] **if** *length y* = *1* **and** *y* ! *0* **for** *y*
using *that* **by** (*metis* (*full-types*) *One-nat-def Suc-length-conv length-0-conv nth-Cons′*)
ultimately have *fverify* ⟨*x*, *u*⟩ = [𝕀]
using *alt*(*5*) **by** *simp*
then show *?lhs*
using *u*(*1*) *alt*(*6*) **by** *auto*
qed
qed
ultimately show *?thesis*
using *that* **by** *simp*
qed

end

Chapter 6

Reducing $\mathcal{NP}$ languages to SAT

theory *Reducing*
imports *Satisfiability Oblivious*
begin

We have already shown that SAT is in $\mathcal{NP}$. It remains to show that SAT is $\mathcal{NP}$-hard, that is, that every language $L \in \mathcal{NP}$ can be polynomial-time reduced to SAT. This, in turn, can be split in two parts. First, showing that for every x there is a CNF formula Φ such that $x \in L$ iff. Φ is satisfiable. Second, that Φ can be computed from x in polynomial time. This chapter is devoted to the first part, which is the core of the proof. In the subsequent two chapters we painstakingly construct a polynomial-time Turing machine computing Φ from x in order to show something that is usually considered "obvious".

The proof corresponds to lemma 2.11 from the textbook [2]. Of course we have to be much more explicit than the textbook, and the first section describes in some detail how we derive the formula Φ.

6.1 Introduction

Let $L \in \mathcal{NP}$. In order to reduce L to SAT, we need to construct for every string $x \in \{\mathbb{O}, \mathbb{I}\}^*$ a CNF formula Φ such that $x \in L$ iff. Φ is satisfiable. In this section we describe how Φ looks like.

6.1.1 Preliminaries

We denote the length of a string $s \in \{\mathbb{O}, \mathbb{I}\}^*$ by $|s|$. We define

$$num(s) = \begin{cases} k & \text{if } s = \mathbb{I}^k \mathbb{O}^{|s|-k}, \\ |s| + 1 & \text{otherwise.} \end{cases}$$

Essentially *num* interprets some strings as unary codes of numbers. All other strings are interpreted as an "error value".

For a string s and a sequence $w \in \mathbb{N}^n$ of numbers we write $s(w)$ for $num(s_{w_0} \dots s_{w_{n-1}})$. Likewise for an assignment $\alpha\colon \mathbb{N} \to \{\mathbb{O}, \mathbb{I}\}$ we write $\alpha(w) = num(\alpha(w_0) \dots \alpha(w_{n-1}))$.

We define two families of CNF formulas. Variables are written $v_0, v_1, v_2, \dots$, and negated variables are written $\bar{v}_0, \bar{v}_1, \bar{v}_2, \dots$ Let $w \in \mathbb{N}^n$ be a list of numbers. For $k \le n$ define

$$\Psi(w, k) = \bigwedge_{i=0}^{k-1} v_{w_i} \wedge \bigwedge_{i=k}^{n-1} \bar{v}_{w_i}.$$

This formula is satisfied by an assignment α iff. $\alpha(w) = k$. In a similar fashion we define for $n > 2$,

$$\Upsilon(w) = v_{w_0} \wedge v_{w_1} \wedge \bigwedge_{i=3}^{n-1} \bar{v}_{w_i},$$

which is satisfied by an assignment α iff. $\alpha(w) \in \{2, 3\} = \{\mathbf{0}, \mathbf{1}\}$, where as usual the boldface $\mathbf{0}$ and $\mathbf{1}$ refer to the symbols represented by the numbers 2 and 3.

For $a \le b$ we write $[a:b]$ for the interval $[a, \dots, b-1] \in \mathbb{N}^{b-a}$. For intervals the CNF formulas become:

$$\Psi([a:b],k) = \bigwedge_{i=a}^{a+k-1} v_i \wedge \bigwedge_{i=a+k}^{b-1} \bar{v}_i \qquad \text{and} \qquad \Upsilon([a:b]) = v_a \wedge v_{a+1} \wedge \bigwedge_{i=a+3}^{b-1} \bar{v}_i.$$

Let φ be a CNF formula and let $\sigma \in \mathbb{N}^*$ be a sequence of variable indices such that for all variables v_i occurring in φ we have $i < |\sigma|$. Then we define the CNF formula $\sigma(\varphi)$ as the formula resulting from replacing every variable v_i in φ by the variable v_{σ_i}. This corresponds to our function *relabel*.

6.1.2 Construction of the CNF formula

Let M be the two-tape oblivious verifier Turing machine for L from lemma *NP-imp-oblivious-2tape*. Let p be the polynomial function for the length of the certificates, and let $T\colon \mathbb{N} \to \mathbb{N}$ be the polynomial running-time bound. Let G be M's alphabet size.

Let $x \in \{\mathbb{O}, \mathbb{I}\}^n$ be fixed thoughout the rest of this section. We seek a CNF formula Φ that is satisfiable iff. $x \in L$. We are going to transform "$x \in L$" via several equivalent statements to the statement "Φ is satisfiable" for a suitable Φ defined along the way. The Isabelle formalization later in this chapter does not prove these equivalences explicitly. They are only meant to explain the shape of Φ.

1st equivalence

From lemma *NP-imp-oblivious-2tape* about M we get the first equivalent statement: There exists a certificate $u \in \{\mathbb{O}, \mathbb{I}\}^{p(n)}$ such that M on input $\langle x, u\rangle$ halts with the symbol $\mathbf{1}$ under its output tape head. The running time of M is bounded by $T(|\langle x, u\rangle|) = T(2n + 2 + 2p(n))$. We abbreviate $|\langle x, u\rangle| = 2n + 2 + 2p(n)$ by m.

2nd equivalence

For the second equivalent statement, we define what the textbook calls "snapshots". For every $u \in \{\mathbb{O}, \mathbb{I}\}^{p(n)}$ let $z_0^u(t)$ be the symbol under the input tape head of M on input $\langle x, u\rangle$ at step t. Similarly we define $z_1^u(t)$ as the symbol under the output tape head of M at step t and $z_2^u(t)$ as the state M is in at step t. A triple $z^u(t) = (z_0^u(t), z_1^u(t), z_2^u(t))$ is called a snapshot. For the initial snapshot we have:

$$z_0^u(0) = z_1^u(0) = \triangleright \qquad \text{and} \qquad z_2^u(0) = 0. \tag{Z0}$$

The crucial idea is that the snapshots for $t > 0$ can be characterized recursively using two auxiliary functions *inputpos* and *prev*.

Since M is oblivious, the positions of the tape heads on input $\langle x, u\rangle$ after t steps are the same for all u of length $p(n)$. We denote the input head positions by $\mathit{inputpos}(t)$.

For every t we denote by $\mathit{prev}(t)$ the last step before t in which the output tape head of M was in the same cell as in step t. Due to M's obliviousness this is again the same for all u of length $p(n)$. If there is no such previous step, because t is the first time the cell is reached, we set $\mathit{prev}(t) = t$. (This deviates from the textbook, which sets $\mathit{prev}(t) = 1$.) In the other case we have $\mathit{prev}(t) < t$.

Also due to M's obliviousness, the halting time on input $\langle x, u\rangle$ is the same for all u of length $p(n)$, and we denote it by $T' \le T(|\langle x, u\rangle|)$. Thus we have $\mathit{inputpos}(t) \le T'$ for all t. If we define the symbol sequence $y(u) = \triangleright\langle x, u\rangle\Box^{T'}$, the first component of the snapshots is, for arbitrary t:

$$z_0^u(t) = y(u)_{\mathit{inputpos}(t)}. \tag{Z1}$$

Next we consider the snapshot components $z_1^u(t)$ for $t > 0$. First consider the case $\mathit{prev}(t) < t$; that is, the last time before t when M's output tape head was in the same cell as in step t was in step $\mathit{prev}(t)$. The snapshot for step $\mathit{prev}(t)$ has exactly the information needed to calculate the actions of M at step t: the symbols read from both tapes and the state which M is in. In some sort of hybrid notation:

$$z_1^u(t) = (M \;!\; z_2^u(\mathit{prev}(t)))\; [z_0^u(\mathit{prev}(t)), z_1^u(\mathit{prev}(t))]\; [.]\; 1. \tag{Z2}$$

In the other case, $\mathit{prev}(t) = t$, the output tape head has not been in this cell before and is thus reading a blank. It cannot be reading the start symbol because the output tape head was in cell zero at step $t = 0$ already. Formally:

$$z_1^u(t) = \Box. \tag{Z3}$$

The state $z_2^u(t)$ for $t > 0$ can be computed from the state $z_2^u(t-1)$ in the previous step and the symbols $z_0^u(t-1)$ and $z_1^u(t-1)$ read in the previous step:

$$z_2^u(t) = \mathit{fst}\ ((M\ !\ z_2^u(t-1))\ [z_0^u(t-1), z_1^u(t-1)]). \tag{Z4}$$

For a string $u \in \{\mathbb{O}, \mathbb{I}\}^{p(n)}$ the equations (Z0) – (Z4) uniquely determine all the $z^u(0), \ldots, z^u(T')$. Conversely, the snapshots for u satisfy all the equations. Therefore the equations characterize the sequence of snapshots.

The condition that M halts with the output tape head on $\mathbf{1}$ can be expressed with snapshots:

$$z_1^u(T') = \mathbf{1}. \tag{Z5}$$

This yields our second equivalent statement: $x \in L$ iff. there is a $u \in \{\mathbb{O}, \mathbb{I}\}^{p(n)}$ and a sequence $z^u(0), \ldots, z^u(T')$ satisfying the equations (Z0) – (Z5).

3rd equivalence

The length of $y(u)$ is $m' := m + 1 + T' = 3 + 2n + 2p(n) + T'$ because $|\langle x, u\rangle| = m$ plus the start symbol plus the T' blanks.

For the next equivalence we observe that the strings $y(u)$ for $u \in \{\mathbb{O}, \mathbb{I}\}^{p(n)}$ can be characterized as follows. Consider a predicate on strings y:

(Y0) $|y| = m'$;

(Y1) $y_0 = \triangleright$ (the start symbol);

(Y2) $y_{2n+1} = y_{2n+2} = \mathbf{1}$ (the separator in the pair encoding);

(Y3) $y_{2i+1} = \mathbf{0}$ for $i = 0, \ldots, n-1$ (the zeros before x);

(Y4) $y_{2n+2+2i+1} = \mathbf{0}$ for $i = 0, \ldots, p(n)-1$ (the zeros before u);

(Y5) $y_{2n+2p(n)+3+i} = \Box$ for $i = 0, \ldots, T'-1$ (the blanks after the input proper);

(Y6) $y_{2i+2} = \begin{cases} \mathbf{0} & \text{if } x_i = \mathbb{O}, \\ \mathbf{1} & \text{otherwise} \end{cases}$ for $i = 0, \ldots, n-1$ (the bits of x);

(Y7) $y_{2n+4+2i} \in \{\mathbf{0}, \mathbf{1}\}$ for $i = 0, \ldots, p(n)-1$ (the bits of u).

Every $y(u)$ for some u of length $p(n)$ satisfies this predicate. Conversely, from a y satisfying the predicate, a u of length $p(n)$ can be extracted such that $y = y(u)$.

From that we get the third equivalent statement: $x \in L$ iff. there is a $y \in \{0, \ldots, G-1\}^{m'}$ with (Y0) – (Y7) and a sequence $z^u(0), \ldots, z^u(T')$ with (Z0) – (Z5).

4th equivalence

Each element of y is a symbol from M's alphabet, that is, a number less than G. The same goes for the first two elements of each snapshot, $z_0^u(t)$ and $z_1^u(t)$. The third element, $z_2^u(t)$, is a number less than or equal to the number of states of M. Let H be the maximum of G and the number of states. Every element of y and of the snapshots can then be represented by a bit string of length H using *num* (the textbook uses binary, but unary is simpler for us). So we use $3H$ bits to represent one snapshot. There are $T' + 1$ snapshots until M halts. Thus all elements of all snapshots can be represented by a string of length $3H \cdot (T' + 1)$. Together with the string of length $N := H \cdot m'$ for the input tape contents y, we have a total length of $N + 3H \cdot (T' + 1)$.

The equivalence can thus be stated as $x \in L$ iff. there is a string $s \in \{\mathbb{O}, \mathbb{I}\}^{N+3H\cdot(T'+1)}$ with certain properties. To write these properties we introduce some intervals:

- $\gamma_i = [iH : (i+1)H]$ for $i < m'$,
- $\zeta_0(t) = [N + 3Ht : N + 3Ht + H]$ for $t \le T'$,
- $\zeta_1(t) = [N + 3Ht + H : N + 3Ht + 2H]$ for $t \le T'$,

- $\zeta_2(t) = [N + 3Ht + 2H : N + 3H(t+1)]$ for $t \le T'$.

These intervals slice the string s in intervals of length H. The string s must satisfy properties analogous to (Y0) – (Y7), which we express using the intervals γ_i:

(Y0) $|s| = N + 3H(T'+1)$

(Y1) $s(\gamma_0) = \triangleright$ (the start symbol);

(Y2) $s(\gamma_{2n+1}) = s(\gamma_{2n+2}) = \mathbf{1}$ (the separator in the pair encoding);

(Y3) $s(\gamma_{2i+1}) = \mathbf{0}$ for $i = 0, \dots, n-1$ (the zeros before x);

(Y4) $s(\gamma_{2n+2+2i+1}) = \mathbf{0}$ for $i = 0, \dots, p(n)-1$ (the zeros before u);

(Y5) $s(\gamma_{2n+2p(n)+3+i}) = \square$ for $i = 0, \dots, T'-1$ (the blanks after the input proper);

(Y6) $s(\gamma_{2i+2}) = \begin{cases} \mathbf{0} & \text{if } x_i = \mathbb{O}, \\ \mathbf{1} & \text{otherwise} \end{cases}$ for $i = 0, \dots, n-1$ (the bits of x);

(Y7) $s(\gamma_{2n+4+2i}) \in \{\mathbf{0}, \mathbf{1}\}$ for $i = 0, \dots, p(n)-1$ (the bits of u).

Moreover the string s must satisfy (Z0) – (Z5). For these properties we use the ζ intervals.

(Z0) $s(\zeta_0(0)) = s(\zeta_1(0)) = \triangleright$ and $s(\zeta_2(0)) = 0$,

(Z1) $s(\zeta_0(t)) = s(\gamma_{inputpos(t)})$ for $t = 1, \dots, T'$,

(Z2) $s(\zeta_1(t)) = (M \; ! \; s(\zeta_2(prev(t))) \; [s(\zeta_0(prev(t))), s(\zeta_1(prev(t)))] \; [.] \; 1$ for $t = 1, \dots, T'$ with $prev(t) < t$,

(Z3) $s(\zeta_1(t)) = \square$ for $t = 1, \dots, T'$ with $prev(t) = t$,

(Z4) $s(\zeta_2(t)) = fst \; ((M \; ! \; s(\zeta_2(t-1)) \; [s(\zeta_0(t-1), s(\zeta_1(t-1)])$ for $t = 1, \dots, T'$,

(Z5) $s(\zeta_1(T')) = \mathbf{1}$.

5th equivalence

An assignment is an infinite bit string. For formulas over variables with indices in the interval $[0 : N + 3H(T'+1)]$, only the initial $N + 3H(T'+1)$ bits of the assignment are relevant. If we had a CNF formula Φ over these variables that is satisfied exactly by the assignments α for which there is an s with the above properties and $\alpha(i) = s_i$ for all $i < |s|$, then the existence of such an s would be equivalent to Φ being satisfiable.

Next we construct such a CNF formula.

Most properties are easy to translate using the formulas Ψ and Υ. For example, $s(\gamma_0) = \triangleright$ corresponds to $\alpha(\gamma_0) = \triangleright$. A formula that is satisfied exactly by assignments with this property is $\Psi(\gamma_0, 1)$. Likewise the property $s(\gamma_{2n+4+2i}) \in \{\mathbf{0}, \mathbf{1}\}$ corresponds to the CNF formula $\Upsilon(\gamma_{2n+4+2i})$.

Property (Y0) corresponds to Φ having only variables $0, \dots, N+3H(T'+1)-1$. The other (Y$\cdot$) properties become:

(Y1) $\Phi_1 := \Psi(\gamma_0, 1)$,

(Y2) $\Phi_2 := \Psi(\gamma_{2n+1}, 3) \wedge \Psi(\gamma_{2n+2}, 3)$,

(Y3) $\Phi_3 := \bigwedge_{i=0}^{n-1} \Psi(\gamma_{2i+1}, 2)$,

(Y4) $\Phi_4 := \bigwedge_{i=0}^{p(n)-1} \Psi(\gamma_{2n+2+2i+1}, 2)$,

(Y5) $\Phi_5 := \bigwedge_{i=0}^{T'-1} \Psi(\gamma_{2n+2p(n)+3+i}, 0)$,

(Y6) $\Phi_6 := \bigwedge_{i=0}^{n-1} \Psi(\gamma_{2i+2}, x_i + 2)$,

(Y7) $\Phi_7 := \bigwedge_{i=0}^{p(n)-1} \Upsilon(\gamma_{2n+4+2i})$.

Property (Z0) and property (Z5) become these formulas:

(Z0) $\Phi_0 := \Psi(\zeta_0(0), 1) \wedge \Psi(\zeta_1(0), 1) \wedge \Psi(\zeta_2(0), 0)$,

(Z5) $\Phi_8 := \Psi(\zeta_1(T'), 3)$.

The remaining properties (Z1) – (Z4) are more complex. They apply to $t = 1, \dots T'$. Let us first consider the case $prev(t) < t$. With α for s the properties become:

(Z1) $\alpha(\zeta_0(t)) = \alpha(\gamma_{inputpos(t)})$,

(Z2) $\alpha(\zeta_1(t)) = ((M \mathbin{!} \alpha(\zeta_2(prev(t)))) \; [\alpha(\zeta_0(prev(t))), \alpha(\zeta_1(prev(t)))]) \; [.] \; 1$,

(Z4) $\alpha(\zeta_2(t)) = \mathit{fst} \; ((M \mathbin{!} \alpha(\zeta_2(t-1))) \; [\alpha(\zeta_0(t-1)), \alpha(\zeta_1(t-1))])$.

For any t the properties (Z1), (Z2), (Z4) use at most $10H$ variable indices, namely all the variable indices in the nine ζ's and in $\gamma_{inputpos(t)}$, each of which have H indices.
Now if the set of all these variable indices was $\{0, \dots, 10H - 1\}$ we could apply lemma *depon-ex-formula* to get a CNF formula ψ over these variables that "captures the spirit" of the properties. We would then merely have to relabel the formula with the actual variable indices we need for each t. More precisely, let $w_i = [iH : (i + 1)H]$ for $i = 0, \dots, 9$ and consider the following criterion for α on the variable indices $\{0, \dots 10H - 1\}$:

(F_1) $\alpha(w_6) = \alpha(w_9)$,

(F_2) $\alpha(w_7) = ((M \mathbin{!} \alpha(w_5)) \; [\alpha(w_3), \alpha(w_4)]) \; [.] \; 1$,

(F_3) $\alpha(w_8) = \mathit{fst} \; ((M \mathbin{!} \alpha(w_2)) \; [\alpha(w_0), \alpha(w_1)])$.

Lemma *depon-ex-formula* gives us a formula ψ satisfied exactly by those assignments α that meet the conditions (F_1), (F_2), (F_3). From this ψ we can create all the formulas we need for representing the properties (Z1), (Z2), (Z4) by substituting ("relabeling" in our terminology) the variables $[0, 10H)$ appropriately. The substitution for step t is

$$\varrho_t = \zeta_0(t-1) \circ \zeta_1(t-1) \circ \zeta_2(t-1) \circ \zeta_0(prev(t)) \circ \zeta_1(prev(t)) \circ \zeta_2(prev(t)) \circ \zeta_0(t) \circ \zeta_1(t) \circ \zeta_2(t) \circ \gamma_{inputpos(t)},$$

where $\circ$ denotes the concatenation of lists. Then $\varrho_t(\psi)$ is CNF formula satisfied exactly by the assignments α satisfying (Z1), (Z2), (Z4).
For the case $prev(t) = t$ we have a criterion on the variable indices $\{0, \dots, 7H - 1\}$:

(F'_1) $\alpha(w_3) = \alpha(w_6)$,

(F'_2) $\alpha(w_4) = \Box$,

(F'_3) $\alpha(w_5) = \mathit{fst} \; ((M \mathbin{!} \alpha(w_2)) \; [\alpha(w_0), \alpha(w_1)])$,

whence lemma *depon-ex-formula* supplies us with a formula ψ'. With appropriate substitutions

$$\varrho'_t = \zeta_0(t-1) \circ \zeta_1(t-1) \circ \zeta_2(t-1) \circ \zeta_0(t) \circ \zeta_1(t) \circ \zeta_2(t) \circ \gamma_{inputpos(t)},$$

we then define CNF formulas χ_t for all $t = 1, \dots, T'$:

$$\chi_t = \begin{cases} \varrho_t(\psi) & \text{if } prev(t) < t, \\ \varrho'_t(\psi') & \text{if } prev(t) = t. \end{cases}$$

The point of all that is that we can hard-code ψ and ψ' in the TM performing the reduction (to be built in the final chapter) and for each t the TM only needs to construct the substitution ϱ_t or ϱ'_t and perform the relabeling. Turing machines that perform these operations will be constructed in the next chapter.
Since all χ_t are in CNF, so is the conjunction

$$\Phi_9 := \bigwedge_{t=1}^{T'} \chi_t \; .$$

Finally the complete CNF formula is:

$$\Phi := \Phi_0 \wedge \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \Phi_4 \wedge \Phi_5 \wedge \Phi_6 \wedge \Phi_7 \wedge \Phi_8 \wedge \Phi_9 \; .$$

6.2 Auxiliary CNF formulas

In this section we define the CNF formula families Ψ and Υ. In the introduction both families were parameterized by intervals of natural numbers. Here we generalize the definition to allow arbitrary sequences of numbers although we will not need this generalization.

The number of variables set to true in a list of variables:

definition *numtrue* :: *assignment* $\Rightarrow$ *nat list* $\Rightarrow$ *nat* **where**
  numtrue α *vs* $\equiv$ *length* (*filter* α *vs*)

Checking whether the list of bits assigned to a list *vs* of variables has the form $\mathbb{I}\ldots\mathbb{I}\mathbb{O}\ldots\mathbb{O}$:

definition *blocky* :: *assignment* $\Rightarrow$ *nat list* $\Rightarrow$ *nat* $\Rightarrow$ *bool* **where**
  blocky α *vs* $k \equiv \forall\, i{<}$*length vs*. α (*vs* ! *i*) $\longleftrightarrow i < k$

The next function represents the notation $\alpha(\gamma)$ from the introduction, albeit generalized to lists that are not intervals γ.

definition *unary* :: *assignment* $\Rightarrow$ *nat list* $\Rightarrow$ *nat* **where**
  unary α *vs* $\equiv$ *if* ($\exists\, k$. *blocky* α *vs k*) *then numtrue* α *vs else Suc* (*length vs*)

lemma *numtrue-remap*:
  assumes $\forall\, s{\in}$*set seq*. $s <$ *length* σ
  shows *numtrue* (*remap* σ α) *seq* = *numtrue* α (*reseq* σ *seq*)
proof −
  have ∗: *length* (*filter f xs*) = *length* (*filter* (*f* ∘ ((!) *xs*)) [*0*..<*length xs*]) **for** *f* **and** *xs* :: *'a list*
    using *length-filter-map map-nth* **by** *metis*

  let *?s-alpha* = *remap* σ α
  let *?s-seq* = *reseq* σ *seq*
  have *numtrue ?s-alpha seq* = *length* (*filter ?s-alpha seq*)
    using *numtrue-def* **by** *simp*
  then have *lhs*: *numtrue ?s-alpha seq* = *length* (*filter* (*?s-alpha* ∘ ((!) *seq*)) [*0*..<*length seq*])
    using ∗ **by** *auto*

  have *numtrue* α *?s-seq* = *length* (*filter* α *?s-seq*)
    using *numtrue-def* **by** *simp*
  then have *numtrue* α *?s-seq* = *length* (*filter* (α ∘ ((!) *?s-seq*)) [*0*..<*length ?s-seq*])
    using ∗ **by** *metis*
  then have *rhs*: *numtrue* α *?s-seq* = *length* (*filter* (α ∘ ((!) *?s-seq*)) [*0*..<*length seq*])
    using *reseq-def* **by** *simp*

  have (*?s-alpha* ∘ ((!) *seq*)) *i* = (α ∘ ((!) *?s-seq*)) *i* **if** *i* < *length seq* **for** *i*
    using *assms remap-def reseq-def that* **by** *simp*
  then show *?thesis*
    using *lhs rhs* **by** (*metis atLeastLessThan-iff filter-cong set-upt*)
qed

lemma *unary-remap*:
  assumes $\forall\, s{\in}$*set seq*. $s <$ *length* σ
  shows *unary* (*remap* σ α) *seq* = *unary* α (*reseq* σ *seq*)
proof −
  have ∗: *blocky* (*remap* σ α) *seq k* = *blocky* α (*reseq* σ *seq*) *k* **for** *k*
    using *blocky-def remap-def reseq-def assms* **by** *simp*
  let *?alpha* = *remap* σ α **and** *?seq* = *reseq* σ *seq*
  show *?thesis*
  proof (*cases* $\exists\, k$. *blocky ?alpha seq k*)
    case *True*
    then show *?thesis*
      using ∗ *unary-def numtrue-remap assms* **by** *simp*
  next
    case *False*
    then have *unary ?alpha seq* = *Suc* (*length seq*)
      using *unary-def* **by** *simp*

```
    moreover have ¬ (∃ k. blocky α ?seq k)
      using False ∗ assms by simp
    ultimately show ?thesis
      using unary-def by simp
  qed
qed
```

Now we define the family Ψ of CNF formulas. It is parameterized by a list *vs* of variable indices and a number $k \leq |vs|$. The formula is satisfied exactly by those assignments that set to true the first k variables in *vs* and to false the other variables. This is more general than we need, because for us *vs* will always be an interval.

```
definition Psi :: nat list ⇒ nat ⇒ formula (‹Ψ›) where
  Ψ vs k ≡ map (λs. [Pos s]) (take k vs) @ map (λs. [Neg s]) (drop k vs)

lemma Psi-unary:
  assumes k ≤ length vs and α ⊨ Ψ vs k
  shows unary α vs = k
proof −
  have α ⊨ map (λs. [Pos s]) (take k vs)
    using satisfies-def assms Psi-def by simp
  then have satisfies-clause α [Pos v] if v ∈ set (take k vs) for v
    using satisfies-def that by simp
  then have satisfies-literal α (Pos v) if v ∈ set (take k vs) for v
    using satisfies-clause-def that by simp
  then have pos: α v if v ∈ set (take k vs) for v
    using that by simp

  have α ⊨ map (λs. [Neg s]) (drop k vs)
    using satisfies-def assms Psi-def by simp
  then have satisfies-clause α [Neg v] if v ∈ set (drop k vs) for v
    using satisfies-def that by simp
  then have satisfies-literal α (Neg v) if v ∈ set (drop k vs) for v
    using satisfies-clause-def that by simp
  then have neg: ¬ α v if v ∈ set (drop k vs) for v
    using that by simp

  have blocky α vs k
  proof −
    have α (vs ! i) if i < k for i
      using pos that assms(1) by (metis in-set-conv-nth length-take min-absorb2 nth-take)
    moreover have ¬ α (vs ! i) if i ≥ k i < length vs for i
      using that assms(1) neg
      by (metis Cons-nth-drop-Suc list.set-intros(1) set-drop-subset-set-drop subsetD)
    ultimately show ?thesis
      using blocky-def by (metis linorder-not-le)
  qed
  moreover have numtrue α vs = k
  proof −
    have filter α vs = take k vs
      using pos neg
     by (metis (mono-tags, lifting) append.right-neutral append-take-drop-id filter-True filter-append filter-empty-conv)
    then show ?thesis
      using numtrue-def assms(1) by simp
  qed
  ultimately show ?thesis
    using unary-def by auto
qed
```

We will only ever consider cases where $k \leq |vs|$. So we can use *blocky* to show that an assignment satisfies a Ψ formula.

```
lemma satisfies-Psi:
  assumes k ≤ length vs and blocky α vs k
  shows α ⊨ Ψ vs k
```

proof −
  have $\alpha \models$ *map* ($\lambda s.$ [*Pos s*]) (*take k vs*)
    (**is** $\alpha \models$ *?phi*)
  proof −
    {
      fix *c* :: *clause*
      assume *c*: $c \in$ *set ?phi*
      then obtain *s* **where** $c =$ [*Pos s*] **and** $s \in$ *set* (*take k vs*)
        by *auto*
      then obtain *i* **where** $i < k$ **and** $s = vs\ !\ i$
        using *assms*(*1*)
        by (*smt* (*verit*, *best*) *in-set-conv-nth le-antisym le-trans nat-le-linear nat-less-le nth-take nth-take-lemma take-all-iff*)
      then have $c =$ [*Pos* (*vs* ! *i*)]
        using ‹$c =$ [*Pos s*]› **by** *simp*
      moreover have $i <$ *length vs*
        using *assms*(*1*) ‹$i < k$› **by** *simp*
      ultimately have α (*vs* ! *i*)
        using *assms*(*2*) *blocky-def* ‹$i < k$› **by** *blast*
      then have *satisfies-clause* α *c*
        using *satisfies-clause-def* **by** (*simp add*: ‹$c =$ [*Pos* (*vs* ! *i*)]›)
    }
    then show *?thesis*
      using *satisfies-def* **by** *simp*
  qed
  moreover have $\alpha \models$ *map* ($\lambda s.$ [*Neg s*]) (*drop k vs*)
    (**is** $\alpha \models$ *?phi*)
  proof −
    {
      fix *c* :: *clause*
      assume *c*: $c \in$ *set ?phi*
      then obtain *s* **where** $c =$ [*Neg s*] **and** $s \in$ *set* (*drop k vs*)
        by *auto*
      then obtain *j* **where** $j <$ *length vs* $- k$ **and** $s =$ *drop k vs* ! *j*
        by (*metis in-set-conv-nth length-drop*)
      define *i* **where** $i = j + k$
      then have $i \geq k$ **and** $s = vs\ !\ i$
        by (*auto simp add*: ‹$s =$ *drop k vs* ! *j*› *add.commute assms*(*1*) *i-def*)
      then have $c =$ [*Neg* (*vs* ! *i*)]
        using ‹$c =$ [*Neg s*]› **by** *simp*
      moreover have $i <$ *length vs*
        using *assms*(*1*) **using** ‹$j <$ *length vs* $- k$› *i-def* **by** *simp*
      ultimately have $\neg\ \alpha$ (*vs* ! *i*)
        using *assms*(*2*) *blocky-def*[*of* α *vs k*] *i-def* **by** *simp*
      then have *satisfies-clause* α *c*
        using *satisfies-clause-def* **by** (*simp add*: ‹$c =$ [*Neg* (*vs* ! *i*)]›)
    }
    then show *?thesis*
      using *satisfies-def* **by** *simp*
  qed
  ultimately show *?thesis*
    using *satisfies-def Psi-def* **by** *auto*
qed

lemma *blocky-imp-unary*:
  assumes $k \leq$ *length vs* **and** *blocky* α *vs k*
  shows *unary* α *vs* $= k$
  using *assms satisfies-Psi Psi-unary* **by** *simp*

The family Υ of CNF formulas also takes as parameter a list of variable indices.

definition *Upsilon* :: *nat list* $\Rightarrow$ *formula* (‹Υ›) **where**
  Υ *vs* $\equiv$ *map* ($\lambda s.$ [*Pos s*]) (*take 2 vs*) @ *map* ($\lambda s.$ [*Neg s*]) (*drop 3 vs*)

For $|vs| > 2$, an assignment satisfies $\Upsilon(vs)$ iff. it satisfies $\Psi(vs, 2)$ or $\Psi(vs, 3)$.

lemma *Psi-2-imp-Upsilon*:
  fixes α :: *assignment*
  assumes $\alpha \models \Psi$ *vs 2* **and** *length vs* > *2*
  shows $\alpha \models \Upsilon$ *vs*
proof −
  have $\alpha \models$ *map* (λ*s.* [*Pos s*]) (*take 2 vs*)
    using *assms Psi-def satisfies-def* **by** *simp*
  moreover have $\alpha \models$ *map* (λ*s.* [*Neg s*]) (*drop 3 vs*)
    using *assms Psi-def satisfies-def*
    by (*smt* (*verit*) *Cons-nth-drop-Suc One-nat-def Suc-1 Un-iff insert-iff list.set*(*2*) *list.simps*(*9*)
      numeral-3-eq-3 set-append)
  ultimately show *?thesis*
    using *Upsilon-def satisfies-def* **by** *auto*
qed

lemma *Psi-3-imp-Upsilon*:
  assumes $\alpha \models \Psi$ *vs 3* **and** *length vs* > *2*
  shows $\alpha \models \Upsilon$ *vs*
proof −
  have $\alpha \models$ *map* (λ*s.* [*Pos s*]) (*take 2 vs*)
    using *assms Psi-def satisfies-def*
    by (*metis eval-nat-numeral*(*3*) *map-append satisfies-append*(*1*) *take-Suc-conv-app-nth*)
  moreover have $\alpha \models$ *map* (λ*s.* [*Neg s*]) (*drop 3 vs*)
    using *assms Psi-def satisfies-def* **by** *simp*
  ultimately show *?thesis*
    using *Upsilon-def satisfies-def* **by** *auto*
qed

lemma *Upsilon-imp-Psi-2-or-3*:
  assumes $\alpha \models \Upsilon$ *vs* **and** *length vs* > *2*
  shows $\alpha \models \Psi$ *vs 2* $\vee$ $\alpha \models \Psi$ *vs 3*
proof −
  have $\alpha \models$ *map* (λ*s.* [*Pos s*]) (*take 2 vs*)
    using *satisfies-def assms Upsilon-def* **by** *simp*
  then have *satisfies-clause* α [*Pos v*] **if** $v \in$ *set* (*take 2 vs*) **for** *v*
    using *satisfies-def that* **by** *simp*
  then have *satisfies-literal* α (*Pos v*) **if** $v \in$ *set* (*take 2 vs*) **for** *v*
    using *satisfies-clause-def that* **by** *simp*
  then have *1*: α *v* **if** $v \in$ *set* (*take 2 vs*) **for** *v*
    using *that* **by** *simp*
  then have *2*: *satisfies-clause* α [*Pos v*] **if** $v \in$ *set* (*take 2 vs*) **for** *v*
    using *that satisfies-clause-def* **by** *simp*

  have $\alpha \models$ *map* (λ*s.* [*Neg s*]) (*drop 3 vs*)
    using *satisfies-def assms Upsilon-def* **by** *simp*
  then have *satisfies-clause* α [*Neg v*] **if** $v \in$ *set* (*drop 3 vs*) **for** *v*
    using *satisfies-def that* **by** *simp*
  then have *satisfies-literal* α (*Neg v*) **if** $v \in$ *set* (*drop 3 vs*) **for** *v*
    using *satisfies-clause-def that* **by** *simp*
  then have *3*: $\neg$ α *v* **if** $v \in$ *set* (*drop 3 vs*) **for** *v*
    using *that* **by** *simp*
  then have *4*: *satisfies-clause* α [*Neg v*] **if** $v \in$ *set* (*drop 3 vs*) **for** *v*
    using *that satisfies-clause-def* **by** *simp*

  show *?thesis*
  proof (*cases* α (*vs* ! *2*))
    case *True*
    then have α *v* **if** $v \in$ *set* (*take 3 vs*) **for** *v*
      using *that 1 assms*(*2*)
      by (*metis* (*no-types*, *lifting*) *in-set-conv-nth le-simps*(*3*) *length-take less-imp-le-nat linorder-neqE-nat*
        min-absorb2 not-less-eq nth-take numeral-One numeral-plus-numeral plus-1-eq-Suc semiring-norm(*3*))
    then have *satisfies-clause* α [*Pos v*] **if** $v \in$ *set* (*take 3 vs*) **for** *v*
      using *that satisfies-clause-def* **by** *simp*

then have $\alpha \models \Psi$ *vs 3*
using *4 Psi-def satisfies-def* **by** *auto*
then show *?thesis*
by *simp*
next
case *False*
then have $\neg\, \alpha\ v$ **if** $v \in$ *set* (*drop 2 vs*) **for** *v*
using *that 3 assms*(*2*)
by (*metis Cons-nth-drop-Suc numeral-plus-numeral numerals*(*1*) *plus-1-eq-Suc semiring-norm*(*3*) *set-ConsD*)
then have *satisfies-clause* α [*Neg v*] **if** $v \in$ *set* (*drop 2 vs*) **for** *v*
using *that satisfies-clause-def* **by** *simp*
then have $\alpha \models \Psi$ *vs 2*
using *2 Psi-def satisfies-def* **by** *auto*
then show *?thesis*
by *simp*
qed
qed

lemma *Upsilon-unary*:
assumes $\alpha \models \Upsilon$ *vs* **and** *length vs* > 2
shows *unary* α *vs* $= 2 \vee$ *unary* α *vs* $= 3$
using *assms Upsilon-imp-Psi-2-or-3 Psi-unary* **by** *fastforce*

6.3 The functions *inputpos* and *prev*

Sequences of the symbol **0**:

definition *zeros* :: *nat* $\Rightarrow$ *symbol list* **where**
zeros n $\equiv$ *string-to-symbols* (*replicate n* $\mathbb{0}$)

lemma *length-zeros* [*simp*]: *length* (*zeros n*) $=$ *n*
using *zeros-def* **by** *simp*

lemma *bit-symbols-zeros*: *bit-symbols* (*zeros n*)
using *zeros-def* **by** *simp*

lemma *zeros*: *zeros n* $=$ *replicate n* **0**
using *zeros-def* **by** *simp*

The assumptions in the following locale are the conditions that according to lemma *NP-imp-oblivious-2tape* hold for all $\mathcal{NP}$ languages. The construction of Φ will take place inside this locale, which in later chapters will be extended to contain the Turing machine outputting Φ and the correctness proof for this Turing machine.

locale *reduction-sat* $=$
fixes *L* :: *language*
fixes *M* :: *machine*
and *G* :: *nat*
and *T p* :: *nat* $\Rightarrow$ *nat*
assumes *T*: *big-oh-poly T*
assumes *p*: *polynomial p*
assumes *tm-M*: *turing-machine 2 G M*
and *oblivious-M*: *oblivious M*
and *T-halt*: $\bigwedge y$. *bit-symbols y* $\Longrightarrow$ *fst* (*execute M* (*start-config 2 y*) (*T* (*length y*))) $=$ *length M*
and *cert*: $\bigwedge x$.
$x \in L \longleftrightarrow (\exists\, u.$ *length u* $=$ *p* (*length x*) $\wedge$ *execute M* (*start-config 2* $\langle x;\, u\rangle$) (*T* (*length* $\langle x;\, u\rangle$)) $<.>$ *1* $=$ **1**)
begin

The value *H* is an upper bound for the number of states of *M* and the alphabet size of *M*.

definition *H* :: *nat* **where**
$H \equiv$ *max G* (*length M*)

lemma *H-ge-G*: $H \geq G$

using *H-def* **by** *simp*

lemma *H-gr-2*: $H > 2$
using *H-def tm-M turing-machine-def* **by** *auto*

lemma *H-ge-3*: $H \geq 3$
using *H-gr-2* **by** *simp*

lemma *H-ge-length-M*: $H \geq$ *length M*
using *H-def* **by** *simp*

The number of symbols used for encoding one snapshot is $Z = 3H$:

definition *Z* :: *nat* **where**
$Z \equiv 3 * H$

The configuration after running M on input y for t steps:

abbreviation *exc* :: *symbol list* $\Rightarrow$ *nat* $\Rightarrow$ *config* **where**
exc y t $\equiv$ *execute M* (*start-config 2 y*) *t*

The function T is just some polynomial upper bound for the running time. The next function, TT, is the actual running time. Since M is oblivious, its running time depends only on the length of the input. The argument *zeros n* is thus merely a placeholder for an arbitrary symbol sequence of length n.

definition *TT* :: *nat* $\Rightarrow$ *nat* **where**
TT n $\equiv$ *LEAST t. fst* (*exc* (*zeros n*) *t*) = *length M*

lemma *TT*: *fst* (*exc* (*zeros n*) (*TT n*)) = *length M*
proof −
let *?P* = λ*t. fst* (*exc* (*zeros n*) *t*) = *length M*
have *?P* (*T n*)
using *T-halt bit-symbols-zeros length-zeros* **by** *metis*
then show *?thesis*
using *wellorder-Least-lemma*[*of ?P*] *TT-def* **by** *simp*
qed

lemma *TT-le*: *TT n* $\leq$ *T n*
using *wellorder-Least-lemma length-zeros TT-def T-halt*[*OF bit-symbols-zeros*[*of n*]] **by** *fastforce*

lemma *less-TT*: $t <$ *TT n* $\Longrightarrow$ *fst* (*exc* (*zeros n*) *t*) < *length M*
proof −
assume $t <$ *TT n*
then have *fst* (*exc* (*zeros n*) *t*) $\neq$ *length M*
using *TT-def not-less-Least* **by** *auto*
moreover have *fst* (*exc* (*zeros n*) *t*) $\leq$ *length M* **for** *t*
using *tm-M start-config-def turing-machine-execute-states* **by** *auto*
ultimately show *fst* (*exc* (*zeros n*) *t*) < *length M*
using *less-le* **by** *blast*
qed

lemma *oblivious-halt-state*:
assumes *bit-symbols zs*
shows *fst* (*exc zs t*) < *length M* $\longleftrightarrow$ *fst* (*exc* (*zeros* (*length zs*)) *t*) < *length M*
proof −
obtain *e* **where**
e: $\forall$ *zs. bit-symbols zs* $\longrightarrow$ ($\exists$ *tps. trace M* (*start-config 2 zs*) (*e* (*length zs*)) (*length M*, *tps*))
using *oblivious-M oblivious-def* **by** *auto*
let *?es* = *e* (*length zs*)
have $\forall$ *i*<*length ?es. fst* (*exc zs i*) < *length M*
using *trace-def e assms* **by** *simp*
moreover have *fst* (*exc zs* (*length ?es*)) = *length M*
using *trace-def e assms* **by** *auto*
moreover have $\forall$ *i*<*length ?es. fst* (*exc* (*zeros* (*length zs*)) *i*) < *length M*
using *length-zeros bit-symbols-zeros trace-def e* **by** *simp*
moreover have *fst* (*exc* (*zeros* (*length zs*)) (*length ?es*)) = *length M*

using *length-zeros bit-symbols-zeros trace-def e assms*
by (*smt* (*verit*, *ccfv-SIG*) *fst-conv*)
ultimately show *?thesis*
by (*metis* (*no-types*, *lifting*) *execute-after-halting-ge le-less-linear*)
qed

corollary *less-TT′*:
assumes *bit-symbols zs* **and** $t < TT$ (*length zs*)
shows *fst* (*exc zs t*) < *length M*
using *assms oblivious-halt-state less-TT* **by** *simp*

corollary *TT′*:
assumes *bit-symbols zs*
shows *fst* (*exc zs* (*TT* (*length zs*))) = *length M*
using *assms TT oblivious-halt-state*
by (*metis* (*no-types*, *lifting*) *fst-conv start-config-def start-config-length less-le tm-M turing-machine-execute-states zero-le zero-less-numeral*)

lemma *exc-TT-eq-exc-T*:
assumes *bit-symbols zs*
shows *exc zs* (*TT* (*length zs*)) = *exc zs* (*T* (*length zs*))
using *execute-after-halting-ge*[*OF TT′*[*OF assms*] *TT-le*] **by** *simp*

The position of the input tape head of M depends only on the length n of the input and the step t, at least as long as the input is over the alphabet $\{\mathbf{0}, \mathbf{1}\}$.

definition *inputpos* :: *nat* ⇒ *nat* ⇒ *nat* **where**
inputpos n t ≡ *exc* (*zeros n*) *t* <#> *0*

lemma *inputpos-oblivious*:
assumes *bit-symbols zs*
shows *exc zs t* <#> *0* = *inputpos* (*length zs*) *t*
proof −
obtain *e* **where**
e: (∀ *zs*. *bit-symbols zs* ⟶ (∃ *tps*. *trace M* (*start-config 2 zs*) (*e* (*length zs*)) (*length M*, *tps*)))
using *oblivious-M oblivious-def* **by** *auto*
let *?es* = *e* (*length zs*)
obtain *tps* **where** *t1*: *trace M* (*start-config 2 zs*) *?es* (*length M*, *tps*)
using *e assms* **by** *auto*
let *?zs* = (*replicate* (*length zs*) *2*)
have *proper-symbols ?zs*
by *simp*
moreover have *length ?zs* = *length zs*
by *simp*
ultimately obtain *tps0* **where** *t0*: *trace M* (*start-config 2 ?zs*) *?es* (*length M*, *tps0*)
using *e* **by** *fastforce*
have *le*: *exc zs t* <#> *0* = *inputpos* (*length zs*) *t* **if** *t* ≤ *length ?es* **for** *t*
proof (*cases t* = *0*)
case *True*
then show *?thesis*
by (*simp add*: *start-config-def inputpos-def*)
next
case *False*
then obtain *i* **where** *i*: *Suc i* = *t*
using *gr0-implies-Suc* **by** *auto*
then have *exc zs* (*Suc i*) <#> *0* = *fst* (*?es* ! *i*)
using *t1 False that Suc-le-lessD trace-def* **by** *auto*
moreover have *exc ?zs* (*Suc i*) <#> *0* = *fst* (*?es* ! *i*)
using *t0 False i that Suc-le-lessD trace-def* **by** *auto*
ultimately show *?thesis*
using *i inputpos-def zeros* **by** *simp*
qed
moreover have *exc zs t* <#> *0* = *inputpos* (*length zs*) *t* **if** *t* > *length ?es*
proof −

have *exc ?zs* (*length ?es*) = (*length M*, *tps0*)
using *t0 trace-def* **by** *simp*
then have ∗: *exc ?zs t* = *exc ?zs* (*length ?es*)
using *that* **by** (*metis execute-after-halting-ge fst-eqD less-or-eq-imp-le*)
have *exc zs* (*length ?es*) = (*length M*, *tps*)
using *t1 trace-def* **by** *simp*
then have *exc zs t* = *exc zs* (*length ?es*)
using *that* **by** (*metis execute-after-halting-ge fst-eqD less-or-eq-imp-le*)
then show *?thesis*
using ∗ *le*[*of length ?es*] **by** (*simp add*: *inputpos-def zeros*)
qed
ultimately show *?thesis*
by *fastforce*
qed

The position of the tape head on the output tape of M also depends only on the length n of the input and the step t.

lemma *oblivious-headpos-1*:
assumes *bit-symbols zs*
shows *exc zs t* <#> *1* = *exc* (*zeros* (*length zs*)) *t* <#> *1*
proof −
obtain *e* **where**
e: (∀ *zs. bit-symbols zs* ⟶ (∃ *tps. trace M* (*start-config 2 zs*) (*e* (*length zs*)) (*length M*, *tps*)))
using *oblivious-M oblivious-def* **by** *auto*
let *?es* = *e* (*length zs*)
obtain *tps* **where** *t1*: *trace M* (*start-config 2 zs*) *?es* (*length M*, *tps*)
using *e assms* **by** *auto*
let *?zs* = (*replicate* (*length zs*) *2*)
have *proper-symbols ?zs*
by *simp*
moreover have *length ?zs* = *length zs*
by *simp*
ultimately obtain *tps0* **where** *t0*: *trace M* (*start-config 2 ?zs*) *?es* (*length M*, *tps0*)
using *e* **by** *fastforce*
have *le*: *exc zs t* <#> *1* = *exc* (*zeros* (*length zs*)) *t* <#> *1* **if** *t* ≤ *length ?es* **for** *t*
proof (*cases t* = *0*)
case *True*
then show *?thesis*
by (*simp add*: *start-config-def inputpos-def*)
next
case *False*
then obtain *i* **where** *i*: *Suc i* = *t*
using *gr0-implies-Suc* **by** *auto*
then have *exc zs* (*Suc i*) <#> *1* = *snd* (*?es* ! *i*)
using *t1 False that Suc-le-lessD trace-def* **by** *auto*
moreover have *exc ?zs* (*Suc i*) <#> *1* = *snd* (*?es* ! *i*)
using *t0 False i that Suc-le-lessD trace-def* **by** *auto*
ultimately show *?thesis*
using *i inputpos-def zeros* **by** *simp*
qed
moreover have *exc zs t* <#> *1* = *exc* (*zeros* (*length zs*)) *t* <#> *1* **if** *t* > *length ?es*
proof −
have *exc ?zs* (*length ?es*) = (*length M*, *tps0*)
using *t0 trace-def* **by** *simp*
then have ∗: *exc ?zs t* = *exc ?zs* (*length ?es*)
using *that* **by** (*metis execute-after-halting-ge fst-eqD less-or-eq-imp-le*)
have *exc zs* (*length ?es*) = (*length M*, *tps*)
using *t1 trace-def* **by** *simp*
then have *exc zs t* = *exc zs* (*length ?es*)
using *that* **by** (*metis execute-after-halting-ge fst-eqD less-or-eq-imp-le*)
then show *?thesis*
using ∗ *le*[*of length ?es*] **by** (*simp add*: *inputpos-def zeros*)
qed

ultimately show *?thesis*
  using *le-less-linear* **by** *blast*
qed

The value *prev*(t) is the most recent step in which M's output tape head was in the same position as in step t. If no such step exists, *prev*(t) is set to t. Again due to M being oblivious, *prev* depends only on the length n of the input (and on t, of course).

definition *prev* :: *nat* ⇒ *nat* ⇒ *nat* **where**
  prev n t ≡
    if ∃ *t'<t. exc* (*zeros n*) *t'* <#> *1* = *exc* (*zeros n*) *t* <#> *1*
    then GREATEST t'. t' < *t* ∧ *exc* (*zeros n*) *t'* <#> *1* = *exc* (*zeros n*) *t* <#> *1*
    else t

lemma *oblivious-prev*:
  assumes *bit-symbols zs*
  shows *prev* (*length zs*) *t* =
   (*if* ∃ *t'<t. exc zs t'* <#> *1* = *exc zs t* <#> *1*
    then GREATEST t'. t' < *t* ∧ *exc zs t'* <#> *1* = *exc zs t* <#> *1*
    else t)
  using *prev-def assms oblivious-headpos-1* **by** *auto*

lemma *prev-less*:
  assumes ∃ *t'<t. exc* (*zeros n*) *t'* <#> *1* = *exc* (*zeros n*) *t* <#> *1*
  shows *prev n t* < *t* ∧ *exc* (*zeros n*) (*prev n t*) <#> *1* = *exc* (*zeros n*) *t* <#> *1*
proof −
  let *?P* = λ*t'. t'* < *t* ∧ *exc* (*zeros n*) *t'* <#> *1* = *exc* (*zeros n*) *t* <#> *1*
  have *prev n t* = (*GREATEST t'. t'* < *t* ∧ *exc* (*zeros n*) *t'* <#> *1* = *exc* (*zeros n*) *t* <#> *1*)
    using *assms prev-def* **by** *simp*
  moreover have ∀ *y. ?P y* ⟶ *y* ≤ *t*
    by *simp*
  ultimately show *?thesis*
    using *GreatestI-ex-nat*[*OF assms*, *of t*] **by** *simp*
qed

corollary *prev-less'*:
  assumes *bit-symbols zs*
  assumes ∃ *t'<t. exc zs t'* <#> *1* = *exc zs t* <#> *1*
  shows *prev* (*length zs*) *t* < *t* ∧ *exc zs* (*prev* (*length zs*) *t*) <#> *1* = *exc zs t* <#> *1*
  using *prev-less oblivious-headpos-1 assms* **by** *simp*

lemma *prev-greatest*:
  assumes *t'* < *t* **and** *exc* (*zeros n*) *t'* <#> *1* = *exc* (*zeros n*) *t* <#> *1*
  shows *t'* ≤ *prev n t*
proof −
  let *?P* = λ*t'. t'* < *t* ∧ *exc* (*zeros n*) *t'* <#> *1* = *exc* (*zeros n*) *t* <#> *1*
  have *prev n t* = (*GREATEST t'. t'* < *t* ∧ *exc* (*zeros n*) *t'* <#> *1* = *exc* (*zeros n*) *t* <#> *1*)
    using *assms prev-def* **by** *auto*
  moreover have *?P t'*
    using *assms* **by** *simp*
  moreover have ∀ *y. ?P y* ⟶ *y* ≤ *t*
    by *simp*
  ultimately show *?thesis*
    using *Greatest-le-nat*[*of ?P t' t*] **by** *simp*
qed

corollary *prev-greatest'*:
  assumes *bit-symbols zs*
  assumes *t'* < *t* **and** *exc zs t'* <#> *1* = *exc zs t* <#> *1*
  shows *t'* ≤ *prev* (*length zs*) *t*
  using *prev-greatest oblivious-headpos-1 assms* **by** *simp*

lemma *prev-eq*: *prev n t* = *t* ⟷ ¬ (∃ *t'<t. exc* (*zeros n*) *t'* <#> *1* = *exc* (*zeros n*) *t* <#> *1*)
  using *prev-def nat-less-le prev-less* **by** *simp*

```
lemma prev-le: prev n t ≤ t
  using prev-eq prev-less by (metis less-or-eq-imp-le)

corollary prev-eq':
  assumes bit-symbols zs
  shows prev (length zs) t = t ⟷ ¬ (∃t'<t. exc zs t' <#> 1 = exc zs t <#> 1)
  using prev-eq oblivious-headpos-1 assms by simp

lemma prev-between:
  assumes prev n t < t' and t' < t
  shows exc (zeros n) t' <#> 1 ≠ exc (zeros n) (prev n t) <#> 1
  using assms by (metis (no-types, lifting) leD prev-eq prev-greatest prev-less)

lemma prev-write-read:
  assumes bit-symbols zs and n = length zs
    and prev n t < t and cfg = exc zs (prev n t) and t ≤ TT n
  shows exc zs t <.> 1 = (M ! (fst cfg)) [cfg <.> 0, cfg <.> 1] [.] 1
proof -
  let ?cfg = exc zs (Suc (prev n t))
  let ?sas = (M ! (fst cfg)) [cfg <.> 0, cfg <.> 1]
  let ?i = cfg <#> 1
  have 1: fst cfg < length M
    using assms less-TT' by simp
  have 2: ||cfg|| = 2
    using assms execute-num-tapes start-config-length tm-M by auto
  then have 3: read (snd cfg) = [cfg <.> 0, cfg <.> 1]
    using read-def
    by (smt (verit) Cons-eq-map-conv Suc-1 length-0-conv length-Suc-conv list.simps(8)
      nth-Cons-0 nth-Cons-Suc numeral-1-eq-Suc-0 numeral-One)

  have *: (?cfg <:> 1) ?i = (M ! (fst cfg)) [cfg <.> 0, cfg <.> 1] [.] 1
  proof -
    have ?cfg <!> 1 = exe M cfg <!> 1
      by (simp add: assms)
    also have ... = sem (M ! (fst cfg)) cfg <!> 1
      using 1 exe-lt-length by simp
    also have ... = act (snd ((M ! (fst cfg)) (read (snd cfg))) ! 1) (snd cfg ! 1)
      using sem-snd-tm tm-M 1 2 by (metis Suc-1 lessI prod.collapse)
    also have ... = act (?sas [!] 1) (cfg <!> 1)
      using 3 by simp
    finally have *: ?cfg <!> 1 = act (?sas [!] 1) (cfg <!> 1) .

    have (?cfg <:> 1) ?i = fst (?cfg <!> 1) ?i
      by simp
    also have ***: ... = ((fst (cfg <!> 1))(?i := (?sas [.] 1))) ?i
      using * act by simp
    also have ... = ?sas [.] 1
      by simp
    finally show (?cfg <:> 1) ?i = ?sas [.] 1
      using *** by simp
  qed

  have ((exc zs t') <:> 1) ?i = (M ! (fst cfg)) [cfg <.> 0, cfg <.> 1] [.] 1
    if Suc (prev n t) ≤ t' and t' ≤ t for t'
  using that
  proof (induction t' rule: nat-induct-at-least)
    case base
    then show ?case
      using * by simp
  next
    case (Suc m)
    let ?cfg-m = exc zs m
```

from *Suc* **have** *between*: *prev n t < m m < t*
by *simp-all*
then have ∗: *?cfg-m <#> 1* ≠ *?i*
using *prev-between assms oblivious-headpos-1* **by** *simp*

have *m < TT n*
using *Suc assms* **by** *simp*
then have *1*: *fst ?cfg-m < length M*
using *assms less-TT′* **by** *simp*
have *2*: ||*?cfg-m*|| = *2*
using *execute-num-tapes start-config-length tm-M* **by** *auto*

have *exc zs* (*Suc m*) *<!> 1 = exe M ?cfg-m <!> 1*
by *simp*
also have ... = *sem* (*M* ! (*fst ?cfg-m*)) *?cfg-m <!> 1*
using *1 exe-lt-length* **by** *simp*
also have ... = *act* (*snd* ((*M* ! (*fst ?cfg-m*)) (*read* (*snd ?cfg-m*))) ! *1*) (*snd ?cfg-m* ! *1*)
using *sem-snd-tm tm-M 1 2* **by** (*metis Suc-1 lessI prod.collapse*)
finally have *exc zs* (*Suc m*) *<!> 1 = act* (*snd* ((*M* ! (*fst ?cfg-m*)) (*read* (*snd ?cfg-m*))) ! *1*) (*snd ?cfg-m* ! *1*) **.**
then have (*exc zs* (*Suc m*) *<:> 1*) *?i = fst* (*snd ?cfg-m* ! *1*) *?i*
using ∗ *act-changes-at-most-pos′* **by** *simp*
then show *?case*
using *Suc* **by** *simp*
qed
then have ((*exc zs t*) *<:> 1*) *?i* = (*M* ! (*fst cfg*)) [*cfg <.> 0*, *cfg <.> 1*] [.] *1*
using *Suc-leI assms* **by** *simp*
moreover have *?i = exc zs t <#> 1*
using *assms*(*1*,*2*,*4*) *oblivious-headpos-1 prev-eq prev-less* **by** (*smt* (*verit*))
ultimately show *?thesis*
by *simp*
qed

lemma *prev-no-write*:
assumes *bit-symbols zs* **and** *n = length zs*
and *prev n t = t* **and** *t* ≤ *TT n* **and** *t > 0*
shows *exc zs t <.> 1* = □
proof −
let *?i = exc zs t <#> 1*

have *1*: ¬ (∃ *t′<t. exc zs t′ <#> 1 = ?i*)
using *prev-eq′ assms*(*1*,*2*,*3*) **by** *simp*

have *2*: *?i > 0*
proof (*rule ccontr*)
assume ¬ *0 < ?i*
then have *eq0*: *?i = 0*
by *simp*
moreover have *exc zs 0 <#> 1 = 0*
by (*simp add*: *start-config-pos*)
ultimately show *False*
using *1 assms*(*5*) **by** *auto*
qed

have *3*: (*exc zs* (*Suc t′*) *<:> 1*) *i* = (*exc zs t′ <:> 1*) *i* **if** *i* ≠ *exc zs t′ <#> 1* **for** *i t′*
proof (*cases fst* (*exc zs t′*) *< length M*)
case *True*
let *?cfg = exc zs t′*
have *len2*: ||*?cfg*|| = *2*
using *execute-num-tapes start-config-length tm-M* **by** *auto*
have *exc zs* (*Suc t′*) *<!> 1 = exe M ?cfg <!> 1*
by *simp*
also have ... = *sem* (*M* ! (*fst ?cfg*)) *?cfg <!> 1*

using *True exe-lt-length* **by** *simp*
also have ... = *act* (*snd* ((*M* ! (*fst ?cfg*)) (*read* (*snd ?cfg*))) ! *1*) (*snd ?cfg* ! *1*)
using *sem-snd-tm tm-M True len2* **by** (*metis Suc-1 lessI prod.collapse*)
finally have *exc zs* (*Suc t'*) <!> *1* = *act* (*snd* ((*M* ! (*fst ?cfg*)) (*read* (*snd ?cfg*))) ! *1*) (*snd ?cfg* ! *1*) **.**
then have (*exc zs* (*Suc t'*) <:> *1*) *i* = *fst* (*snd ?cfg* ! *1*) *i*
using *that act-changes-at-most-pos'* **by** *simp*
then show *?thesis*
by *simp*
next
case *False*
then show *?thesis*
using *that* **by** (*simp add*: *exe-def*)
qed

have (*exc zs t'* <:> *1*) *?i* = (*exc zs 0* <:> *1*) *?i* **if** $t' \le t$ **for** *t'*
using *that*
proof (*induction t'*)
case *0*
then show *?case*
by *simp*
next
case (*Suc t'*)
then show *?case*
by (*metis 1 3 Suc-leD Suc-le-lessD*)
qed
then have *exc zs t* <.> *1* = (*exc zs 0* <:> *1*) *?i*
by *simp*
then show *?thesis*
using *2 One-nat-def execute.simps*(*1*) *start-config1 less-2-cases-iff less-one* **by** *presburger*
qed

The intervals γ_i and $w_0, \ldots, w_9$ do not depend on x, and so can be defined here already.

definition *gamma* :: *nat* ⇒ *nat list* (‹γ›) **where**
γ *i* ≡ [*i* ∗ *H*..<*Suc i* ∗ *H*]

lemma *length-gamma* [*simp*]: *length* (γ *i*) = *H*
using *gamma-def* **by** *simp*

abbreviation w_0 ≡ [*0*..<*H*]
abbreviation w_1 ≡ [*H*..<*2*∗*H*]
abbreviation w_2 ≡ [*2*∗*H*..<*Z*]
abbreviation w_3 ≡ [*Z*..<*Z*+*H*]
abbreviation w_4 ≡ [*Z*+*H*..<*Z*+*2*∗*H*]
abbreviation w_5 ≡ [*Z*+*2*∗*H*..<*2*∗*Z*]
abbreviation w_6 ≡ [*2*∗*Z*..<*2*∗*Z*+*H*]
abbreviation w_7 ≡ [*2*∗*Z*+*H*..<*2*∗*Z*+*2*∗*H*]
abbreviation w_8 ≡ [*2*∗*Z*+*2*∗*H*..<*3*∗*Z*]
abbreviation w_9 ≡ [*3*∗*Z*..<*3*∗*Z*+*H*]

lemma *unary-upt-eq*:
fixes α_1 α_2 :: *assignment*
and *lower upper k* :: *nat*
assumes ∀ *i*<*k*. α_1 *i* = α_2 *i* **and** *upper* ≤ *k*
shows *unary* α_1 [*lower*..<*upper*] = *unary* α_2 [*lower*..<*upper*]
proof −
have *numtrue* α_1 [*lower*..<*upper*] = *numtrue* α_2 [*lower*..<*upper*]
proof −
let *?vs* = [*lower*..<*upper*]
have *filter* α_1 *?vs* = *filter* α_2 *?vs*
using *assms* **by** (*metis atLeastLessThan-iff filter-cong less-le-trans set-upt*)
then show *?thesis*
using *numtrue-def* **by** *simp*
qed

moreover have *blocky* α_1 [*lower*..<*upper*] = *blocky* α_2 [*lower*..<*upper*]
  using *blocky-def assms* **by** *auto*
ultimately show *?thesis*
  using *assms unary-def* **by** *simp*
qed

For the case *prev m t* < *t*, we have the following predicate on assignments, which corresponds to (F_1), (F_2), (F_3) from the introduction:

definition *F* :: *assignment* ⇒ *bool* **where**
  F α ≡
    unary α w_6 = *unary* α w_9 ∧
    unary α w_7 = (*M* ! (*unary* α w_5)) [*unary* α w_3, *unary* α w_4] [.] *1* ∧
    unary α w_8 = *fst* ((*M* ! (*unary* α w_2)) [*unary* α w_0, *unary* α w_1])

lemma *depon-F*: *depon* (*3* ∗ *Z* + *H*) *F*
  using *depon-def F-def Z-def unary-upt-eq* **by** *simp*

There is a CNF formula ψ that contains the first $3Z + H$ variables and is satisfied by exactly the assignments specified by F.

definition *psi* :: *formula* (‹ψ›) **where**
  ψ ≡ *SOME* φ.
    fsize φ ≤ (*3* ∗ *Z* + *H*) ∗ *2* ^ (*3* ∗ *Z* + *H*) ∧
    length φ ≤ *2* ^ (*3* ∗ *Z* + *H*) ∧
    variables φ ⊆ {..<*3* ∗ *Z* + *H*} ∧
    (∀ α. *F* α = α ⊨ φ)

lemma *psi*:
  fsize ψ ≤ (*3* ∗ *Z* + *H*) ∗ *2* ^ (*3*∗*Z* + *H*) ∧
   length ψ ≤ *2* ^ (*3* ∗ *Z* + *H*) ∧
   variables ψ ⊆ {..<*3* ∗ *Z* + *H*} ∧
   (∀ α. *F* α = α ⊨ ψ)
  using *psi-def someI-ex*[*OF depon-ex-formula*[*OF depon-F*]] **by** *metis*

lemma *satisfies-psi*:
  assumes *length* σ = *3* ∗ *Z* + *H*
  shows α ⊨ *relabel* σ ψ = *remap* σ α ⊨ ψ
  using *assms psi satisfies-sigma* **by** *simp*

lemma *psi-F*: *remap* σ α ⊨ ψ = *F* (*remap* σ α)
  using *psi* **by** *simp*

corollary *satisfies-F*:
  assumes *length* σ = *3* ∗ *Z* + *H*
  shows α ⊨ *relabel* σ ψ = *F* (*remap* σ α)
  using *assms satisfies-psi psi-F* **by** *simp*

For the case *prev m t* = *t*, the following predicate corresponds to (F'_1), (F'_2), (F'_3) from the introduction:

definition *F'* :: *assignment* ⇒ *bool* **where**
  F' α ≡
    unary α w_3 = *unary* α w_6 ∧
    unary α w_4 = *0* ∧
    unary α w_5 = *fst* ((*M* ! (*unary* α w_2)) [*unary* α w_0, *unary* α w_1])

lemma *depon-F'*: *depon* (*2* ∗ *Z* + *H*) *F'*
  using *depon-def F'-def Z-def unary-upt-eq* **by** *simp*

The CNF formula ψ' is analogous to ψ from the previous case.

definition *psi'* :: *formula* (‹ψ''›) **where**
  ψ' ≡ *SOME* φ.
    fsize φ ≤ (*2*∗*Z*+*H*) ∗ *2* ^ (*2*∗*Z*+*H*) ∧
    length φ ≤ *2* ^ (*2*∗*Z*+*H*) ∧
    variables φ ⊆ {..<*2*∗*Z*+*H*} ∧

$(\forall\, \alpha.\ F'\ \alpha = \alpha \models \varphi)$

lemma *psi*$'$:
  $fsize\ \psi' \leq (2{*}Z{+}H) * 2 \mathbin{\hat{}} (2{*}Z{+}H) \wedge$
  $length\ \psi' \leq 2 \mathbin{\hat{}} (2{*}Z{+}H) \wedge$
  $variables\ \psi' \subseteq \{..{<}2{*}Z{+}H\} \wedge$
  $(\forall\, \alpha.\ F'\ \alpha = \alpha \models \psi')$
  using *psi*$'$*-def someI-ex*[*OF depon-ex-formula*[*OF depon-F*$'$]] **by** *metis*

lemma *satisfies-psi*$'$:
  assumes $length\ \sigma = 2{*}Z{+}H$
  shows $\alpha \models relabel\ \sigma\ \psi' = remap\ \sigma\ \alpha \models \psi'$
  using *assms psi*$'$ *satisfies-sigma* **by** *simp*

lemma *psi*$'$*-F*$'$: $remap\ \sigma\ \alpha \models \psi' = F'\ (remap\ \sigma\ \alpha)$
  using *psi*$'$ **by** *simp*

corollary *satisfies-F*$'$:
  assumes $length\ \sigma = 2{*}Z{+}H$
  shows $\alpha \models relabel\ \sigma\ \psi' = F'\ (remap\ \sigma\ \alpha)$
  using *assms satisfies-psi*$'$ *psi*$'$*-F*$'$ **by** *simp*

end

6.4 Snapshots

The snapshots and much of the rest of the construction of Φ depend on the string x. We encapsulate this in a sublocale of *reduction-sat*.

locale *reduction-sat-x* = *reduction-sat* +
  fixes x :: *string*
begin

abbreviation n :: *nat* **where**
  $n \equiv length\ x$

Turing machines consume the string x as a sequence xs of symbols:

abbreviation xs :: *symbol list* **where**
  $xs \equiv string\text{-}to\text{-}symbols\ x$

lemma *bs-xs*: *bit-symbols xs*
  by *simp*

For the verifier Turing machine M we are only concerned with inputs of the form $\langle x, u\rangle$ for a string u of length $p(n)$. The pair $\langle x, u\rangle$ has the length $m = 2n + 2p(n) + 2$.

definition m :: *nat* **where**
  $m \equiv 2 * n + 2 * p\ n + 2$

On input $\langle x, u\rangle$ the Turing machine M halts after $T' = TT(m)$ steps.

definition T' :: *nat* **where**
  $T' \equiv TT\ m$

The positions of both of M's tape heads are bounded by T'.

lemma *inputpos-less*: $inputpos\ m\ t \leq T'$
proof $-$
  define u :: *string* **where** $u = replicate\ (p\ n)\ False$
  let $?i = inputpos\ m\ t$
  have y: $bit\text{-}symbols\ \langle x;\ u\rangle$
    by *simp*
  have *len*: $length\ \langle x;\ u\rangle = m$
    using *u-def m-def length-pair* **by** *simp*
  then have $exc\ \langle x;\ u\rangle\ t <\#> 0 \leq T'$

    using *TT*′[*OF y*] *T*′*-def head-pos-le-halting-time*[*OF tm-M*, *of* ⟨*x*; *u*⟩ *T*′ *0*] **by** *simp*
  then show *?thesis*
    using *inputpos-oblivious*[*OF y*] *len* **by** *simp*
qed

lemma *headpos-1-less*: *exc* (*zeros m*) *t* <#> *1* ≤ *T*′
proof −
  define *u* :: *string* **where** *u* = *replicate* (*p n*) *False*
  let *?i* = *inputpos m t*
  have *y*: *bit-symbols* ⟨*x*; *u*⟩
    by *simp*
  have *len*: *length* ⟨*x*; *u*⟩ = *m*
    using *u-def m-def length-pair* **by** *simp*
  then have *exc* ⟨*x*; *u*⟩ *t* <#> *1* ≤ *T*′
    using *TT*′[*OF y*] *T*′*-def head-pos-le-halting-time*[*OF tm-M*, *of* ⟨*x*; *u*⟩ *T*′ *1*] **by** *simp*
  then show *?thesis*
    using *oblivious-headpos-1*[*OF y*] *len* **by** *simp*
qed

The formula Φ must contain a condition for every symbol that M is reading from the input tape. While T' is an upper bound for the input tape head position of M, it might be that T' is less than the length of the input $\langle x, u\rangle$. So the portion of the input read by M might be a prefix of the input or it might be the input followed by some blanks afterwards. This would make for an awkward case distinction. We do not have to be very precise here and can afford to bound the portion of the input tape read by M by the number $m' = 2n + 2p(n) + 3 + T'$, which is the length of the start symbol followed by the input $\langle x, u\rangle$ followed by T' blanks. This symbol sequence was called $y(u)$ in the introduction. Here we will call it *ysymbols u*.

definition *m*′ :: *nat* **where**
  m′ ≡ *2* ∗ *n* + *2* ∗ *p n* + *3* + *T*′

definition *ysymbols* :: *string* ⇒ *symbol list* **where**
  ysymbols u ≡ *1* # ⟨*x*; *u*⟩ @ *replicate T*′ *0*

lemma *length-ysymbols*: *length u* = *p n* ⟹ *length* (*ysymbols u*) = *m*′
  using *ysymbols-def m*′*-def m-def length-pair* **by** *simp*

lemma *ysymbols-init*:
  assumes *i* < *length* (*ysymbols u*)
  shows *ysymbols u* ! *i* = (*start-config 2* ⟨*x*; *u*⟩ <:> *0*) *i*
proof −
  let *?y* = ⟨*x*; *u*⟩
  have *init*: *start-config 2 ?y* <:> *0* = (λ*i*. *if i* = *0 then 1 else if i* ≤ *length ?y then ?y* ! (*i* − *1*) *else 0*)
    using *start-config-def* **by** *auto*
  have *i* < *length ?y* + *1* + *T*′
    using *assms ysymbols-def* **by** *simp*
  then consider *i* = *0* | *i* > *0* ∧ *i* ≤ *length ?y* | *i* > *length ?y* ∧ *i* < *length ?y* + *1* + *T*′
    by *linarith*
  then show *ysymbols u* ! *i* = (*start-config 2 ?y* <:> *0*) *i*
  proof (*cases*)
    case *1*
    then show *?thesis*
      using *ysymbols-def init* **by** *simp*
  next
    case *2*
    then have *ysymbols u* ! *i* = ⟨*x*; *u*⟩ ! (*i* − *1*)
      using *ysymbols-def*
        by (*smt* (*verit*, *del-insts*) *Nat.add-diff-assoc diff-is-0-eq gr-zeroI le-add-diff-inverse le-add-diff-inverse2 less-numeral-extra*(*1*) *nat-le-linear nth-Cons-pos nth-append zero-less-diff*)
    then show *?thesis*
      using *init 2* **by** *simp*
  next
    case *3*
    then have (*start-config 2 ?y* <:> *0*) *i* = *0*

```
      using init by simp
    moreover have ysymbols u ! i = 0
      unfolding ysymbols-def using 3 nth-append[of 1#⟨x; u⟩ replicate T′ 0 i] by auto
    ultimately show ?thesis
      by simp
  qed
qed

lemma ysymbols-at-0: ysymbols u ! 0 = 1
  using ysymbols-def by simp

lemma ysymbols-first-at:
  assumes j < length x
  shows ysymbols u ! (2∗j+1) = 2
    and ysymbols u ! (2∗j+2) = (if x ! j then 3 else 2)
proof −
  have ∗: ysymbols u = (1 # ⟨x; u⟩) @ replicate T′ 0
    using ysymbols-def by simp

  let ?i = 2 ∗ j + 1
  have len: 2∗j < length ⟨x, u⟩
    using assms length-string-pair by simp
  have ?i < length (1 # ⟨x; u⟩)
    using assms length-pair by simp
  then have ysymbols u ! ?i = (1 # ⟨x; u⟩) ! ?i
    using ∗ nth-append by metis
  also have ... = ⟨x; u⟩ ! (2∗j)
    by simp
  also have ... = 2
    using string-pair-first-nth assms len by simp
  finally show ysymbols u ! ?i = 2 .

  let ?i = 2 ∗ j + 2
  have len2: ?i < length (1 # ⟨x; u⟩)
    using assms length-pair by simp
  then have ysymbols u ! ?i = (1 # ⟨x; u⟩) ! ?i
    using ∗ nth-append by metis
  also have ... = ⟨x; u⟩ ! (2∗j+1)
    by simp
  also have ... = (if x ! j then 3 else 2)
    using string-pair-first-nth(2) assms len2 by simp
  finally show ysymbols u ! ?i = (if x ! j then 3 else 2) .
qed

lemma ysymbols-at-2n1: ysymbols u ! (2∗n+1) = 3
proof −
  let ?i = 2 ∗ n + 1
  have ysymbols u ! ?i = ⟨x; u⟩ ! (2∗n)
    using ysymbols-def
    by (metis (no-types, lifting) add.commute add-2-eq-Suc′ le-add2 le-imp-less-Suc length-pair
      less-SucI nth-Cons-Suc nth-append plus-1-eq-Suc)
  also have ... = (if ⟨x, u⟩ ! (2∗n) then 3 else 2)
    using length-pair by simp
  also have ... = 3
    using string-pair-sep-nth by simp
  finally show ?thesis .
qed

lemma ysymbols-at-2n2: ysymbols u ! (2∗n+2) = 3
proof −
  let ?i = 2 ∗ n + 2
  have ysymbols u ! ?i = ⟨x; u⟩ ! (2∗n+1)
    by (simp add: ysymbols-def)
```

(*smt* (*verit*, *del-insts*) *add.right-neutral add-2-eq-Suc′ length-greater-0-conv length-pair lessI less-add-same-cancel1 less-trans-Suc list.size*(*3*) *mult-0-right mult-pos-pos nth-append zero-less-numeral*)
also have ... = (*if* ⟨*x*, *u*⟩ ! (*2*∗*n*+*1*) *then 3 else 2*)
using *length-pair* **by** *simp*
also have ... = *3*
using *string-pair-sep-nth* **by** *simp*
finally show *?thesis* **.**
qed

lemma *ysymbols-second-at*:
assumes *j* < *length u*
shows *ysymbols u* ! (*2*∗*n*+*2*+*2*∗*j*+*1*) = *2*
and *ysymbols u* ! (*2*∗*n*+*2*+*2*∗*j*+*2*) = (*if u* ! *j then 3 else 2*)
proof −
have ∗: *ysymbols u* = (*1* # ⟨*x*; *u*⟩) @ *replicate T′ 0*
using *ysymbols-def* **by** *simp*

let *?i* = *2* ∗ *n* + *2* + *2* ∗ *j* + *1*
have *len*: *2* ∗ *n* + *2* + *2*∗*j* < *length* ⟨*x*, *u*⟩
using *assms length-string-pair* **by** *simp*
have *?i* < *length* (*1* # ⟨*x*; *u*⟩)
using *assms length-pair* **by** *simp*
then have *ysymbols u* ! *?i* = (*1* # ⟨*x*; *u*⟩) ! *?i*
using ∗ *nth-append* **by** *metis*
also have ... = ⟨*x*; *u*⟩ ! (*2*∗*n*+*2*+*2*∗*j*)
by *simp*
also have ... = *2*
using *string-pair-second-nth*(*1*) *assms len* **by** *simp*
finally show *ysymbols u* ! *?i* = *2* **.**

let *?i* = *2*∗*n*+*2*+*2* ∗ *j* + *2*
have *len2*: *?i* < *length* (*1* # ⟨*x*; *u*⟩)
using *assms length-pair* **by** *simp*
then have *ysymbols u* ! *?i* = (*1* # ⟨*x*; *u*⟩) ! *?i*
using ∗ *nth-append* **by** *metis*
also have ... = ⟨*x*; *u*⟩ ! (*2*∗*n*+*2*+*2*∗*j*+*1*)
by *simp*
also have ... = (*if u* ! *j then 3 else 2*)
using *string-pair-second-nth*(*2*) *assms len2* **by** *simp*
finally show *ysymbols u* ! *?i* = (*if u* ! *j then 3 else 2*) **.**
qed

lemma *ysymbols-last*:
assumes *length u* = *p n* **and** *i* > *m* **and** *i* < *m* + *1* + *T′*
shows *ysymbols u* ! *i* = *0*
using *assms length-ysymbols m-def m′-def ysymbols-def nth-append*[*of 1*#⟨*x*; *u*⟩ *replicate T′ 0 i*] **by** *simp*

The number of symbols used for unary encoding m' symbols will be called N:

definition *N* :: *nat* **where**
N ≡ *H* ∗ *m′*

lemma *N-eq*: *N* = *H* ∗ (*2* ∗ *n* + *2* ∗ *p n* + *3* + *T′*)
using *m′-def N-def* **by** *simp*

lemma *m′*: *m′* ∗ *H* = *N*
using *m′-def N-def* **by** *simp*

lemma *inputpos-less′*: *inputpos m t* < *m′*
using *inputpos-less m-def m′-def*
by (*metis Suc-1 add-less-mono1 le-neq-implies-less lessI linorder-neqE-nat not-add-less2 numeral-plus-numeral semiring-norm*(*3*) *trans-less-add2*)

lemma *T′-less*: $T' < N$
proof −
  have $T' < 2 * n + 2 * p\ n + 3 + T'$
    by *simp*
  also have ... $< H * (2 * n + 2 * p\ n + 3 + T')$
    using *H-gr-2* **by** *simp*
  also have ... $= N$
    using *N-eq* **by** *simp*
  finally show *?thesis* **.**
qed

The three components of a snapshot:

definition *z0* :: *string* ⇒ *nat* ⇒ *symbol* **where**
  z0 u t ≡ *exc* ⟨*x*; *u*⟩ *t* <.> *0*

definition *z1* :: *string* ⇒ *nat* ⇒ *symbol* **where**
  z1 u t ≡ *exc* ⟨*x*; *u*⟩ *t* <.> *1*

definition *z2* :: *string* ⇒ *nat* ⇒ *state* **where**
  z2 u t ≡ *fst* (*exc* ⟨*x*; *u*⟩ *t*)

lemma *z0-le*: *z0 u t* ≤ *H*
  using *z0-def H-ge-G tape-alphabet*[*OF tm-M*, **where** *?j=0* **and** *?zs*=⟨*x*; *u*⟩] *symbols-lt-pair*[*of x u*] *tm-M turing-machine-def*
  by (*metis* (*no-types*, *lifting*) *dual-order.strict-trans1 less-or-eq-imp-le zero-less-numeral*)

lemma *z1-le*: *z1 u t* ≤ *H*
  using *z1-def H-ge-G tape-alphabet*[*OF tm-M*, **where** *?j=1* **and** *?zs*=⟨*x*; *u*⟩] *symbols-lt-pair*[*of x u*] *tm-M turing-machine-def*
  by (*metis* (*no-types*, *lifting*) *dual-order.strict-trans1 less-or-eq-imp-le one-less-numeral-iff semiring-norm*(*76*))

lemma *z2-le*: *z2 u t* ≤ *H*
proof −
  have *z2 u t* ≤ *length M*
    using *z2-def turing-machine-execute-states*[*OF tm-M*] *start-config-def* **by** *simp*
  then show *?thesis*
    using *H-ge-length-M* **by** *simp*
qed

The next lemma corresponds to (Z1) from the second equivalence mentioned in the introduction. It expresses the first element of a snapshot in terms of $y(u)$ and *inputpos*.

lemma *z0*:
  assumes *length u* = *p n*
  shows *z0 u t* = *ysymbols u* ! (*inputpos m t*)
proof −
  let *?i* = *inputpos m t*
  let *?y* = ⟨*x*; *u*⟩
  have *bit-symbols ?y*
    by *simp*
  have *len*: *length ?y* = *m*
    using *assms m-def length-pair* **by** *simp*

  have *?i* < *length* (*ysymbols u*)
    using *inputpos-less′ assms length-ysymbols* **by** *simp*
  then have ∗: *ysymbols u* ! *?i* = (*start-config 2 ?y* <:> *0*) *?i*
    using *ysymbols-init* **by** *simp*

  have *z0 u t* = *exc ?y t* <.> *0*
    using *z0-def* **by** *simp*
  also have ... = (*start-config 2 ?y* <:> *0*) (*exc ?y t* <#> *0*)
    using *tm-M input-tape-constant start-config-length* **by** *simp*
  also have ... = (*start-config 2 ?y* <:> *0*) *?i*
    using *inputpos-oblivious*[*OF* ‹*bit-symbols ?y*›] *len* **by** *simp*

also have ... = *ysymbols u* ! *?i*
  using ∗ **by** *simp*
finally show *?thesis* **.**
qed

The next lemma corresponds to (Z2) from the second equivalence mentioned in the introduction. It shows how, in the case *prev*(t) < t, the second component of a snapshot can be expressed recursively using snapshots for earlier steps.

lemma *z1*:
  assumes *length u* = *p n* **and** *prev m t* < *t* **and** *t* ≤ *T'*
  shows *z1 u t* = (*M* ! (*z2 u* (*prev m t*))) [*z0 u* (*prev m t*), *z1 u* (*prev m t*)] [.] *1*
proof −
  let *?y* = ⟨*x*; *u*⟩
  let *?cfg* = *exc ?y* (*prev m t*)
  have *bit-symbols ?y*
    by *simp*
  moreover have *len*: *length ?y* = *m*
    using *assms m-def length-pair* **by** *simp*
  ultimately have *exc ?y t* <.> *1* = (*M* ! (*fst ?cfg*)) [*?cfg* <.> *0*, *?cfg* <.> *1*] [.] *1*
    using *prev-write-read*[*of ?y m t ?cfg*] *assms*(*2*,*3*) *T'-def* **by** *fastforce*
  then show *?thesis*
    using *z0-def z1-def z2-def* **by** *simp*
qed

The next lemma corresponds to (Z3) from the second equivalence mentioned in the introduction. It shows that in the case *prev*(t) = t, the second component of a snapshot equals the blank symbol.

lemma *z1'*:
  assumes *length u* = *p n* **and** *prev m t* = *t* **and** *0* < *t* **and** *t* ≤ *T'*
  shows *z1 u t* = □
proof −
  let *?y* = ⟨*x*; *u*⟩
  let *?cfg* = *exc ?y* (*prev m t*)
  have *bit-symbols ?y*
    by *simp*
  moreover have *len*: *length ?y* = *m*
    using *assms m-def length-pair* **by** *simp*
  ultimately have *exc ?y t* <.> *1* = □
    using *prev-no-write*[*of ?y m t*] *assms T'-def* **by** *fastforce*
  then show *?thesis*
    using *z0-def z1-def z2-def* **by** *simp*
qed

The next lemma corresponds to (Z4) from the second equivalence mentioned in the introduction. It shows how the third component of a snapshot can be expressed recursively using snapshots for earlier steps.

lemma *z2*:
  assumes *length u* = *p n* **and** *t* < *T'*
  shows *z2 u* (*Suc t*) = *fst* ((*M* ! (*z2 u t*)) [*z0 u t*, *z1 u t*])
proof −
  let *?y* = ⟨*x*; *u*⟩
  have *bit-symbols ?y*
    by *simp*
  moreover have *len*: *length ?y* = *m*
    using *assms m-def length-pair* **by** *simp*
  ultimately have *run*: *fst* (*exc ?y t*) < *length M*
    using *less-TT' assms*(*2*) *T'-def* **by** *simp*

  have ||*exc ?y t*|| = *2*
    using *start-config-length execute-num-tapes tm-M* **by** *simp*
  then have *snd* (*exc ?y t*) = [*exc ?y t* <!> *0*, *exc ?y t* <!> *1*]
    by *auto* (*smt* (*verit*) *Suc-length-conv length-0-conv nth-Cons-0 nth-Cons-Suc numeral-2-eq-2*)
  then have ∗: *read* (*snd* (*exc ?y t*)) = [*exc ?y t* <.> *0*, *exc ?y t* <.> *1*]
    using *read-def* **by** (*metis* (*no-types*, *lifting*) *list.simps*(*8*) *list.simps*(*9*))

have *z2 u* (*Suc t*) = *fst* (*exc ?y* (*Suc t*))
using *z2-def* **by** *simp*
also have ... = *fst* (*exe M* (*exc ?y t*))
by *simp*
also have ... = *fst* (*sem* (*M* ! *fst* (*exc ?y t*)) (*exc ?y t*))
using *exe-lt-length run* **by** *simp*
also have ... = *fst* (*sem* (*M* ! (*z2 u t*)) (*exc ?y t*))
using *z2-def* **by** *simp*
also have ... = *fst* ((*M* ! (*z2 u t*)) (*read* (*snd* (*exc ?y t*))))
using *sem-fst* **by** *simp*
also have ... = *fst* ((*M* ! (*z2 u t*)) [*exc ?y t* <.> *0*, *exc ?y t* <.> *1*])
using ∗ **by** *simp*
also have ... = *fst* ((*M* ! (*z2 u t*)) [*z0 u t*, *z1 u t*])
using *z0-def z1-def* **by** *simp*
finally show *?thesis* **.**
qed

corollary *z2'*:
assumes *length u* = *p n* **and** $t > 0$ **and** $t \leq T'$
shows *z2 u t* = *fst* ((*M* ! (*z2 u* (*t* − *1*))) [*z0 u* (*t* − *1*), *z1 u* (*t* − *1*)])
using *assms z2* **by** (*metis Suc-diff-1 Suc-less-eq le-imp-less-Suc*)

The intervals ζ_0, ζ_1, and ζ_2 are long enough for a unary encoding of the three components of a snapshot:

definition *zeta0* :: *nat* ⇒ *nat list* (‹ζ_0›) **where**
ζ_0 *t* ≡ [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *H*]

definition *zeta1* :: *nat* ⇒ *nat list* (‹ζ_1›) **where**
ζ_1 *t* ≡ [*N* + *t* ∗ *Z* + *H*..<*N* + *t* ∗ *Z* + *2* ∗ *H*]

definition *zeta2* :: *nat* ⇒ *nat list* (‹ζ_2›) **where**
ζ_2 *t* ≡ [*N* + *t* ∗ *Z* + *2* ∗ *H*..<*N* + (*Suc t*) ∗ *Z*]

lemma *length-zeta0* [*simp*]: *length* (ζ_0 *t*) = *H*
using *zeta0-def* **by** *simp*

lemma *length-zeta1* [*simp*]: *length* (ζ_1 *t*) = *H*
using *zeta1-def* **by** *simp*

lemma *length-zeta2* [*simp*]: *length* (ζ_2 *t*) = *H*
using *zeta2-def Z-def* **by** *simp*

The substitutions ϱ_t, which have to be applied to ψ to get the CNF formulas χ_t for the case $prev(t) < t$:

definition *rho* :: *nat* ⇒ *nat list* (‹ϱ›) **where**
ϱ *t* ≡
ζ_0 (*t* − *1*) @ ζ_1 (*t* − *1*) @ ζ_2 (*t* − *1*) @
ζ_0 (*prev m t*) @ ζ_1 (*prev m t*) @ ζ_2 (*prev m t*) @
ζ_0 *t* @ ζ_1 *t* @ ζ_2 *t* @
γ (*inputpos m t*)

lemma *length-rho*: *length* (ϱ *t*) = *3* ∗ *Z* + *H*
using *rho-def Z-def* **by** *simp*

The substitutions ϱ'_t, which have to be applied to ψ' to get the CNF formulas χ_t for the case $prev(t) = t$:

definition *rho'* :: *nat* ⇒ *nat list* (‹ϱ''›) **where**
ϱ' *t* ≡
ζ_0 (*t* − *1*) @ ζ_1 (*t* − *1*) @ ζ_2 (*t* − *1*) @
ζ_0 *t* @ ζ_1 *t* @ ζ_2 *t* @
γ (*inputpos m t*)

lemma *length-rho'*: *length* (ϱ' *t*) = *2* ∗ *Z* + *H*
using *rho'-def Z-def* **by** *simp*

An auxiliary lemma for accessing the n-th element of a list sandwiched between two lists. It will be applied to $xs = \varrho_t$ or $xs = \varrho'_t$:

lemma *nth-append3*:
  fixes *xs ys zs ws* :: *'a list* **and** *n i* :: *nat*
  assumes *xs* = *ys* @ *zs* @ *ws* **and** *i* < *length zs* **and** *n* = *length ys*
  shows *xs* ! (*n* + *i*) = *zs* ! *i*
  using *assms* **by** (*simp add*: *nth-append*)

The formulas χ_t representing snapshots for $0 < t \leq T'$:

definition *chi* :: *nat* ⇒ *formula* (‹χ›) **where**
  χ *t* ≡ *if prev m t* < *t then relabel* (ϱ *t*) ψ *else relabel* (ϱ′ *t*) ψ′

The crucial feature of the formulas χ_t for $t > 0$ is that they are satisfied by exactly those assignments that represent in their bits N to $N + Z \cdot (T' + 1)$ the $T' + 1$ snapshots of M on input $\langle x, u\rangle$ when the relevant portion of the input tape is encoded in the first N bits of the assignment.

This works because the χ_t constrain the assignment to meet the recursive characterizations (Z1) — (Z4) for the snapshots.

The next two lemmas make this more precise. We first consider the case $prev(t) < t$. The following lemma says α satisfies χ_t iff. α satisfies the properties (Z1), (Z2), and (Z4).

lemma *satisfies-chi-less*:
  fixes α :: *assignment*
  assumes *prev m t* < *t*
  shows α ⊨ χ *t* ⟷
    unary α (ζ_0 *t*) = *unary* α (γ (*inputpos m t*)) ∧
    unary α (ζ_1 *t*) = (*M* ! (*unary* α (ζ_2 (*prev m t*)))) [*unary* α (ζ_0 (*prev m t*)), *unary* α (ζ_1 (*prev m t*))] [.] *1* ∧
    unary α (ζ_2 *t*) = *fst* ((*M* ! (*unary* α (ζ_2 (*t* − *1*)))) [*unary* α (ζ_0 (*t* − *1*)), *unary* α (ζ_1 (*t* − *1*))])
proof −
  let *?sigma* = ϱ *t*
  have α ⊨ χ *t* = α ⊨ *relabel ?sigma psi*
    using *assms chi-def* **by** *simp*
  then have α ⊨ χ *t* = *F* (*remap ?sigma* α)
      (**is** - = *F ?alpha*)
    by (*simp add*: *length-rho satisfies-F*)
  then have ∗: α ⊨ χ *t* =
      (*unary ?alpha* w_6 = *unary ?alpha* w_9 ∧
      unary ?alpha w_7 = (*M* ! (*unary ?alpha* w_5)) [*unary ?alpha* w_3, *unary ?alpha* w_4] [.] *1* ∧
      unary ?alpha w_8 = *fst* ((*M* ! (*unary ?alpha* w_2)) [*unary ?alpha* w_0, *unary ?alpha* w_1]))
    using *F-def* **by** *simp*

  have *w-less-len-rho*:
    ∀ *s*∈*set* w_0. *s* < *length* (ϱ *t*)
    ∀ *s*∈*set* w_1. *s* < *length* (ϱ *t*)
    ∀ *s*∈*set* w_2. *s* < *length* (ϱ *t*)
    ∀ *s*∈*set* w_3. *s* < *length* (ϱ *t*)
    ∀ *s*∈*set* w_4. *s* < *length* (ϱ *t*)
    ∀ *s*∈*set* w_5. *s* < *length* (ϱ *t*)
    ∀ *s*∈*set* w_6. *s* < *length* (ϱ *t*)
    ∀ *s*∈*set* w_7. *s* < *length* (ϱ *t*)
    ∀ *s*∈*set* w_8. *s* < *length* (ϱ *t*)
    ∀ *s*∈*set* w_9. *s* < *length* (ϱ *t*)
    using *length-rho Z-def* **by** *simp-all*

  have ∗∗: α ⊨ χ *t* =
     (*unary* α (*reseq ?sigma* w_6) = *unary* α (*reseq ?sigma* w_9) ∧
      unary α (*reseq ?sigma* w_7) = (*M* ! (*unary* α (*reseq ?sigma* w_5))) [*unary* α (*reseq ?sigma* w_3), *unary* α (*reseq ?sigma* w_4)] [.] *1* ∧
      unary α (*reseq ?sigma* w_8) = *fst* ((*M* ! (*unary* α (*reseq ?sigma* w_2))) [*unary* α (*reseq ?sigma* w_0), *unary* α (*reseq ?sigma* w_1)]))
    using ∗ *w-less-len-rho unary-remap*[**where** *?σ*=*?sigma* **and** *?α*=α]
    by *presburger*

  have *1*: *reseq ?sigma* w_0 = ζ_0 (*t* − *1*) (**is** *?l* = *?r*)

proof (*rule nth-equalityI*)
show *length ?l = length ?r*
using *zeta0-def* **by** *simp*
show *?l ! i = ?r ! i* **if** *i < length ?l* **for** *i*
proof −
have *1*: *?sigma* = [] @ ζ_0 (*t* − *1*) @ (ζ_1 (*t* − *1*) @ ζ_2 (*t* − *1*) @
ζ_0 (*prev m t*) @ ζ_1 (*prev m t*) @ ζ_2 (*prev m t*) @ ζ_0 *t* @
ζ_1 *t* @ ζ_2 *t* @ γ (*inputpos m t*))
using *rho-def* **by** *simp*
have *?sigma* ! *i* = ζ_0 (*t* − *1*) ! *i*
using *nth-append3*[*OF 1*, *of i 0*] *Z-def that* **by** *simp*
then show *?thesis*
using *reseq-def that* **by** *simp*
qed
qed

have *2*: *reseq ?sigma* w_1 = ζ_1 (*t* − *1*) (**is** *?l = ?r*)
proof (*rule nth-equalityI*)
show *length ?l = length ?r*
using *zeta1-def* **by** *simp*
show *?l ! i = ?r ! i* **if** *i < length ?l* **for** *i*
proof −
have *1*: *?sigma* = ζ_0 (*t* − *1*) @ ζ_1 (*t* − *1*) @ ζ_2 (*t* − *1*) @
ζ_0 (*prev m t*) @ ζ_1 (*prev m t*) @ ζ_2 (*prev m t*) @ ζ_0 *t* @
ζ_1 *t* @ ζ_2 *t* @ γ (*inputpos m t*)
using *rho-def* **by** *simp*
have *?sigma* ! (*H*+*i*) = ζ_1 (*t* − *1*) ! *i*
using *that zeta0-def zeta1-def* **by** (*intro nth-append3*[*OF 1*]) *auto*
then show *?thesis*
using *reseq-def that* **by** *simp*
qed
qed

have *3*: *reseq ?sigma* w_2 = ζ_2 (*t* − *1*) (**is** *?l = ?r*)
proof (*rule nth-equalityI*)
show *len*: *length ?l = length ?r*
using *zeta2-def* **by** *simp*
show *?l ! i = ?r ! i* **if** *i < length ?l* **for** *i*
proof −
have *1*: *?sigma* = (ζ_0 (*t* − *1*) @ ζ_1 (*t* − *1*)) @ ζ_2 (*t* − *1*) @
ζ_0 (*prev m t*) @ ζ_1 (*prev m t*) @ ζ_2 (*prev m t*) @ ζ_0 *t* @
ζ_1 *t* @ ζ_2 *t* @ γ (*inputpos m t*)
using *rho-def* **by** *simp*
have *?sigma* ! (*2*∗*H*+*i*) = ζ_2 (*t* − *1*) ! *i*
using *len that zeta0-def zeta1-def* **by** (*intro nth-append3*[*OF 1*]) *auto*
then show *?thesis*
using *reseq-def that* **by** *simp*
qed
qed

have *4*: *reseq ?sigma* w_3 = ζ_0 (*prev m t*) (**is** *?l = ?r*)
proof (*rule nth-equalityI*)
show *length ?l = length ?r*
using *zeta0-def* **by** *simp*
show *?l ! i = ?r ! i* **if** *i < length ?l* **for** *i*
proof −
have *1*: *?sigma* = (ζ_0 (*t* − *1*) @ ζ_1 (*t* − *1*) @ ζ_2 (*t* − *1*)) @
ζ_0 (*prev m t*) @ ζ_1 (*prev m t*) @ ζ_2 (*prev m t*) @ ζ_0 *t* @
ζ_1 *t* @ ζ_2 *t* @ γ (*inputpos m t*)
using *rho-def* **by** *simp*
have *?sigma* ! (*Z*+*i*) = ζ_0 (*prev m t*) ! *i*
using *that Z-def* **by** (*intro nth-append3*[*OF 1*]) *auto*
then show *?thesis*

    using *reseq-def that* **by** *simp*
  qed
qed

have *5*: *reseq ?sigma* w_4 = ζ_1 (*prev m t*) (**is** *?l* = *?r*)
proof (*rule nth-equalityI*)
  show *length ?l* = *length ?r*
    using *zeta1-def* **by** *simp*
  show *?l* ! *i* = *?r* ! *i* **if** *i* < *length ?l* **for** *i*
  proof −
    have *1*: *?sigma* = (ζ_0 (*t* − *1*) @ ζ_1 (*t* − *1*) @ ζ_2 (*t* − *1*) @
        ζ_0 (*prev m t*)) @ ζ_1 (*prev m t*) @ ζ_2 (*prev m t*) @ ζ_0 *t* @
       ζ_1 *t* @ ζ_2 *t* @ γ (*inputpos m t*)
      using *rho-def* **by** *simp*
    have *?sigma* ! (*Z*+*H*+*i*) = ζ_1 (*prev m t*) ! *i*
      using *that Z-def* **by** (*intro nth-append3*[*OF 1*]) *auto*
    then show *?thesis*
      using *reseq-def that* **by** *simp*
  qed
qed

have *6*: *reseq ?sigma* w_5 = ζ_2 (*prev m t*) (**is** *?l* = *?r*)
proof (*rule nth-equalityI*)
  show *length ?l* = *length ?r*
    using *zeta2-def* **by** *simp*
  show *?l* ! *i* = *?r* ! *i* **if** *i* < *length ?l* **for** *i*
  proof −
    have *1*: *?sigma* = (ζ_0 (*t* − *1*) @ ζ_1 (*t* − *1*) @ ζ_2 (*t* − *1*) @
        ζ_0 (*prev m t*) @ ζ_1 (*prev m t*)) @ ζ_2 (*prev m t*) @ ζ_0 *t* @
       ζ_1 *t* @ ζ_2 *t* @ γ (*inputpos m t*)
      using *rho-def* **by** *simp*
    have *?sigma* ! (*Z*+*2*∗*H*+*i*) = ζ_2 (*prev m t*) ! *i*
      using *that zeta0-def zeta1-def zeta2-def* **by** (*intro nth-append3*[*OF 1*]) *auto*
    then show *?thesis*
      using *reseq-def that* **by** *simp*
  qed
qed

have *7*: *reseq ?sigma* w_6 = ζ_0 *t* (**is** *?l* = *?r*)
proof (*rule nth-equalityI*)
  show *length ?l* = *length ?r*
    using *zeta0-def* **by** *simp*
  show *?l* ! *i* = *?r* ! *i* **if** *i* < *length ?l* **for** *i*
  proof −
    have *1*: *?sigma* = (ζ_0 (*t* − *1*) @ ζ_1 (*t* − *1*) @ ζ_2 (*t* − *1*) @
        ζ_0 (*prev m t*) @ ζ_1 (*prev m t*) @ ζ_2 (*prev m t*)) @
        ζ_0 *t* @
       ζ_1 *t* @ ζ_2 *t* @ γ (*inputpos m t*)
      using *rho-def* **by** *simp*
    have *?sigma* ! (*2*∗*Z*+*i*) = ζ_0 *t* ! *i*
      using *that Z-def* **by** (*intro nth-append3*[*OF 1*]) *auto*
    then show *?thesis*
      using *reseq-def that* **by** *simp*
  qed
qed

have *8*: *reseq ?sigma* w_7 = ζ_1 *t* (**is** *?l* = *?r*)
proof (*rule nth-equalityI*)
  show *length ?l* = *length ?r*
    using *zeta1-def* **by** *simp*
  show *?l* ! *i* = *?r* ! *i* **if** *i* < *length ?l* **for** *i*
  proof −
    have *1*: *?sigma* = (ζ_0 (*t* − *1*) @ ζ_1 (*t* − *1*) @ ζ_2 (*t* − *1*) @

```
        ζ₀ (prev m t) @ ζ₁ (prev m t) @ ζ₂ (prev m t) @ ζ₀ t) @
        ζ₁ t @
        ζ₂ t @ γ (inputpos m t)
      using rho-def by simp
    have ?sigma ! (2*Z+H+i) = ζ₁ t ! i
      using that Z-def by (intro nth-append3[OF 1]) auto
    then show ?thesis
      using reseq-def that by simp
  qed
qed

have 9: reseq ?sigma w₈ = ζ₂ t (is ?l = ?r)
proof (rule nth-equalityI)
  show length ?l = length ?r
    using zeta2-def by simp
  show ?l ! i = ?r ! i if i < length ?l for i
  proof −
    have 1: ?sigma = (ζ₀ (t − 1) @ ζ₁ (t − 1) @ ζ₂ (t − 1) @
        ζ₀ (prev m t) @ ζ₁ (prev m t) @ ζ₂ (prev m t) @ ζ₀ t @ ζ₁ t) @
        ζ₂ t @
        γ (inputpos m t)
      using rho-def by simp
    have ?sigma ! (2*Z+2*H+i) = ζ₂ t ! i
      using that zeta2-def by (intro nth-append3[OF 1]) auto
    then show ?thesis
      using reseq-def that by simp
  qed
qed

have 10: reseq ?sigma w₉ = γ (inputpos m t) (is ?l = ?r)
proof (rule nth-equalityI)
  show length ?l = length ?r
    using gamma-def by simp
  show ?l ! i = ?r ! i if i < length ?l for i
  proof −
    have 1: ?sigma = (ζ₀ (t − 1) @ ζ₁ (t − 1) @ ζ₂ (t − 1) @
        ζ₀ (prev m t) @ ζ₁ (prev m t) @ ζ₂ (prev m t) @
        ζ₀ t @
       ζ₁ t @ ζ₂ t) @ γ (inputpos m t) @ []
      using rho-def by simp
    have ?sigma ! (3*Z+i) = γ (inputpos m t) ! i
      using that Z-def by (intro nth-append3[OF 1]) auto
    then show ?thesis
      using reseq-def that by simp
  qed
qed

show ?thesis
  using ** 1 2 3 4 5 6 7 8 9 10 by simp
qed
```

Next we consider the case $prev(t) = t$. The following lemma says α satisfies χ_t iff. α satisfies the properties (Z1), (Z2), and (Z3).

```
lemma satisfies-chi-eq:
  assumes prev m t = t and t ≤ T'
  shows α ⊨ χ t ⟷
    unary α (ζ₀ t) = unary α (γ (inputpos m t)) ∧
    unary α (ζ₁ t) = 0 ∧
    unary α (ζ₂ t) = fst ((M ! (unary α (ζ₂ (t − 1)))) [unary α (ζ₀ (t − 1)), unary α (ζ₁ (t − 1))])
proof −
  let ?sigma = ϱ' t
  have α ⊨ χ t = α ⊨ relabel ?sigma ψ'
    using assms(1) chi-def by simp
```

then have $\alpha \models \chi\ t = F'\ (remap\ ?sigma\ \alpha)$
  (**is** - $= F'\ ?alpha$)
 by (*simp add*: *length-rho' satisfies-F'*)
then have $*$: $\alpha \models \chi\ t =$
  $(unary\ ?alpha\ w_3 = unary\ ?alpha\ w_6\ \wedge$
  $unary\ ?alpha\ w_4 = 0\ \wedge$
  $unary\ ?alpha\ w_5 = fst\ ((M\ !\ (unary\ ?alpha\ w_2))\ [unary\ ?alpha\ w_0,\ unary\ ?alpha\ w_1]))$
 using *F'-def* **by** *simp*

have *w-less-len-rho'*:
 $\forall s \in set\ w_0.\ s < length\ (\varrho'\ t)$
 $\forall s \in set\ w_1.\ s < length\ (\varrho'\ t)$
 $\forall s \in set\ w_2.\ s < length\ (\varrho'\ t)$
 $\forall s \in set\ w_3.\ s < length\ (\varrho'\ t)$
 $\forall s \in set\ w_4.\ s < length\ (\varrho'\ t)$
 $\forall s \in set\ w_5.\ s < length\ (\varrho'\ t)$
 $\forall s \in set\ w_6.\ s < length\ (\varrho'\ t)$
 using *length-rho' Z-def* **by** *simp-all*

have $**$: $\alpha \models \chi\ t =$
  $(unary\ \alpha\ (reseq\ ?sigma\ w_3) = unary\ \alpha\ (reseq\ ?sigma\ w_6)\ \wedge$
  $unary\ \alpha\ (reseq\ ?sigma\ w_4) = 0\ \wedge$
  $unary\ \alpha\ (reseq\ ?sigma\ w_5) = fst\ ((M\ !\ (unary\ \alpha\ (reseq\ ?sigma\ w_2)))\ [unary\ \alpha\ (reseq\ ?sigma\ w_0),\ unary\ \alpha$
$(reseq\ ?sigma\ w_1)]))$
 using $*$ *w-less-len-rho' unary-remap*[**where** $?\sigma = ?sigma$ **and** $?\alpha = \alpha$]
 by *presburger*

have *1*: $reseq\ ?sigma\ w_0 = \zeta_0\ (t - 1)$ (**is** $?l = ?r$)
proof (*rule nth-equalityI*)
 show $length\ ?l = length\ ?r$
  using *zeta0-def* **by** *simp*
 show $?l\ !\ i = ?r\ !\ i$ **if** $i < length\ ?l$ **for** i
 proof −
  have *1*: $?sigma = []\ @\ \zeta_0\ (t - 1)\ @\ (\zeta_1\ (t - 1)\ @\ \zeta_2\ (t - 1)\ @$
   $\zeta_0\ t\ @\ \zeta_1\ t\ @\ \zeta_2\ t\ @\ \gamma\ (inputpos\ m\ t))$
   using *rho'-def* **by** *simp*
  have $?sigma\ !\ i = \zeta_0\ (t - 1)\ !\ i$
   using *nth-append3*[*OF 1*, *of i 0*] *Z-def that* **by** *simp*
  then show *?thesis*
   using *reseq-def that* **by** *simp*
 qed
qed

have *2*: $reseq\ ?sigma\ w_1 = \zeta_1\ (t - 1)$ (**is** $?l = ?r$)
proof (*rule nth-equalityI*)
 show $length\ ?l = length\ ?r$
  using *zeta1-def* **by** *simp*
 show $?l\ !\ i = ?r\ !\ i$ **if** $i < length\ ?l$ **for** i
 proof −
  have *1*: $?sigma = \zeta_0\ (t - 1)\ @\ \zeta_1\ (t - 1)\ @\ \zeta_2\ (t - 1)\ @$
   $\zeta_0\ t\ @\ \zeta_1\ t\ @\ \zeta_2\ t\ @\ \gamma\ (inputpos\ m\ t)$
   using *rho'-def* **by** *simp*
  have $?sigma\ !\ (H+i) = \zeta_1\ (t - 1)\ !\ i$
   using *that zeta0-def zeta1-def* **by** (*intro nth-append3*[*OF 1*]) *auto*
  then show *?thesis*
   using *reseq-def that* **by** *simp*
 qed
qed

have *3*: $reseq\ ?sigma\ w_2 = \zeta_2\ (t - 1)$ (**is** $?l = ?r$)
proof (*rule nth-equalityI*)
 show *len*: $length\ ?l = length\ ?r$
  using *zeta2-def* **by** *simp*

```
  show ?l ! i = ?r ! i if i < length ?l for i
  proof -
    have 1: ?sigma = (ζ₀ (t - 1) @ ζ₁ (t - 1)) @ ζ₂ (t - 1) @
        ζ₀ t @ ζ₁ t @ ζ₂ t @ γ (inputpos m t)
      using rho'-def by simp
    have ?sigma ! (2*H+i) = ζ₂ (t - 1) ! i
      using len that zeta0-def zeta1-def by (intro nth-append3[OF 1]) auto
    then show ?thesis
      using reseq-def that by simp
  qed
qed

have 4: reseq ?sigma w₃ = ζ₀ t (is ?l = ?r)
proof (rule nth-equalityI)
  show length ?l = length ?r
    using zeta0-def by simp
  show ?l ! i = ?r ! i if i < length ?l for i
  proof -
    have 1: ?sigma = (ζ₀ (t - 1) @ ζ₁ (t - 1) @ ζ₂ (t - 1)) @
        ζ₀ t @ ζ₁ t @ ζ₂ t @ γ (inputpos m t)
      using rho'-def by simp
    have ?sigma ! (Z+i) = ζ₀ t ! i
      using that Z-def by (intro nth-append3[OF 1]) auto
    then show ?thesis
      using reseq-def that by simp
  qed
qed

have 5: reseq ?sigma w₄ = ζ₁ t (is ?l = ?r)
proof (rule nth-equalityI)
  show length ?l = length ?r
    using zeta1-def by simp
  show ?l ! i = ?r ! i if i < length ?l for i
  proof -
    have 1: ?sigma = (ζ₀ (t - 1) @ ζ₁ (t - 1) @ ζ₂ (t - 1) @
        ζ₀ t) @ ζ₁ t @ ζ₂ t @ γ (inputpos m t)
      using rho'-def by simp
    have ?sigma ! (Z+H+i) = ζ₁ t ! i
      using that Z-def by (intro nth-append3[OF 1]) auto
    then show ?thesis
      using reseq-def that by simp
  qed
qed

have 6: reseq ?sigma w₅ = ζ₂ t (is ?l = ?r)
proof (rule nth-equalityI)
  show length ?l = length ?r
    using zeta2-def by simp
  show ?l ! i = ?r ! i if i < length ?l for i
  proof -
    have 1: ?sigma = (ζ₀ (t - 1) @ ζ₁ (t - 1) @ ζ₂ (t - 1) @
        ζ₀ t @ ζ₁ t) @ ζ₂ t @ γ (inputpos m t)
      using rho'-def by simp
    have ?sigma ! (Z+2*H+i) = ζ₂ t ! i
      using that zeta0-def zeta1-def zeta2-def by (intro nth-append3[OF 1]) auto
    then show ?thesis
      using reseq-def that by simp
  qed
qed

have 7: reseq ?sigma w₆ = (γ (inputpos m t)) (is ?l = ?r)
proof (rule nth-equalityI)
  show length ?l = length ?r
```

```
      using gamma-def by simp
    show ?l ! i = ?r ! i if i < length ?l for i
    proof -
      have 1: ?sigma = (ζ₀ (t - 1) @ ζ₁ (t - 1) @ ζ₂ (t - 1) @
          ζ₀ t @ ζ₁ t @ ζ₂ t) @ γ (inputpos m t) @ []
        using rho'-def by simp
      have ?sigma ! (2*Z+i) = γ (inputpos m t) ! i
        using that Z-def gamma-def by (intro nth-append3[OF 1]) auto
      then show ?thesis
        using reseq-def that by simp
    qed
  qed

  show ?thesis
    using ** 1 2 3 4 5 6 7 by simp
qed
```

6.5 The CNF formula Φ

We can now formulate all the parts $\Phi_0, \ldots, \Phi_9$ of the complete formula Φ, and thus Φ itself. Representing the snapshot in step 0:

definition *PHI0* :: *formula* (‹Φ_0›) **where**
  $\Phi_0 \equiv \Psi\ (\zeta_0\ 0)\ 1\ @\ \Psi\ (\zeta_1\ 0)\ 1\ @\ \Psi\ (\zeta_2\ 0)\ 0$

The start symbol at the beginning of the input tape:

definition *PHI1* :: *formula* (‹Φ_1›) **where**
  $\Phi_1 \equiv \Psi\ (\gamma\ 0)\ 1$

The separator **11** between x and u:

definition *PHI2* :: *formula* (‹Φ_2›) **where**
  $\Phi_2 \equiv \Psi\ (\gamma\ (2{*}n{+}1))\ 3\ @\ \Psi\ (\gamma\ (2{*}n{+}2))\ 3$

The zeros before the symbols of x:

definition *PHI3* :: *formula* (‹Φ_3›) **where**
  $\Phi_3 \equiv concat\ (map\ (\lambda i.\ \Psi\ (\gamma\ (2{*}i{+}1))\ 2)\ [0..{<}n])$

The zeros before the symbols of u:

definition *PHI4* :: *formula* (‹Φ_4›) **where**
  $\Phi_4 \equiv concat\ (map\ (\lambda i.\ \Psi\ (\gamma\ (2{*}n{+}2{+}2{*}i{+}1))\ 2)\ [0..{<}p\ n])$

The blank symbols after the input $\langle x, u\rangle$:

definition *PHI5* :: *formula* (‹Φ_5›) **where**
  $\Phi_5 \equiv concat\ (map\ (\lambda i.\ \Psi\ (\gamma\ (2{*}n\ +\ 2{*}p\ n\ +\ 3\ +\ i))\ 0)\ [0..{<}T'])$

The symbols of x:

definition *PHI6* :: *formula* (‹Φ_6›) **where**
  $\Phi_6 \equiv concat\ (map\ (\lambda i.\ \Psi\ (\gamma\ (2{*}i{+}2))\ (if\ x\ !\ i\ then\ 3\ else\ 2))\ [0..{<}n])$

Constraining the symbols of u to be from $\{\mathbf{0}, \mathbf{1}\}$:

definition *PHI7* :: *formula* (‹Φ_7›) **where**
  $\Phi_7 \equiv concat\ (map\ (\lambda i.\ \Upsilon\ (\gamma\ (2{*}n{+}4{+}2{*}i)))\ [0..{<}p\ n])$

Reading a **1** in the final step to signal acceptance of $\langle x, u\rangle$:

definition *PHI8* :: *formula* (‹Φ_8›) **where**
  $\Phi_8 \equiv \Psi\ (\zeta_1\ T')\ 3$

The snapshots after the first and before the last:

definition *PHI9* :: *formula* (‹Φ_9›) **where**
  $\Phi_9 \equiv concat\ (map\ (\lambda t.\ \chi\ (Suc\ t))\ [0..{<}T'])$

The complete formula:

definition *PHI* :: *formula* (‹Φ›) **where**
  $\Phi \equiv \Phi_0$ @ Φ_1 @ Φ_2 @ Φ_3 @ Φ_4 @ Φ_5 @ Φ_6 @ Φ_7 @ Φ_8 @ Φ_9

6.6 Correctness of the formula

We have to show that the formula Φ is satisfiable if and only if $x \in L$. There is a subsection for both of the implications. Instead of $x \in L$ we will use the right-hand side of the following equivalence.

lemma *L-iff-ex-u*: $x \in L \longleftrightarrow (\exists u.\ length\ u = p\ n \wedge exc\ \langle x;\ u\rangle\ T' <.> 1 = \mathbf{1})$
proof −
  have $x \in L \longleftrightarrow (\exists u.\ length\ u = p\ (length\ x) \wedge exc\ \langle x;\ u\rangle\ (T\ (length\ \langle x;\ u\rangle)) <.> 1 = \mathbf{1})$
    using *cert* **by** *simp*
  also have ... $\longleftrightarrow (\exists u.\ length\ u = p\ (length\ x) \wedge exc\ \langle x;\ u\rangle\ (TT\ (length\ \langle x;\ u\rangle)) <.> 1 = \mathbf{1})$
  proof −
    have *bit-symbols* $\langle x;\ u\rangle$ **for** *u*
      by *simp*
    then have $exc\ \langle x;\ u\rangle\ (TT\ (length\ \langle x;\ u\rangle)) = exc\ \langle x;\ u\rangle\ (T\ (length\ \langle x;\ u\rangle))$ **for** *u*
      using *exc-TT-eq-exc-T* **by** *blast*
    then show *?thesis*
      by *simp*
  qed
  also have ... $\longleftrightarrow (\exists u.\ length\ u = p\ n \wedge exc\ \langle x;\ u\rangle\ T' <.> 1 = \mathbf{1})$
    using *T′-def length-pair m-def* **by** *auto*
  finally show *?thesis* **.**
qed

6.6.1 Φ satisfiable implies $x \in L$

The proof starts from an assignment α satisfying Φ and shows that there is a string u of length $p(n)$ such that M, on input $\langle x, u\rangle$, halts with the output tape head on the symbol **1**. The overarching idea is that α, by satisfying Φ, encodes a string u and a computation of M on u that results in M halting with the output tape head on the symbol **1**.

The assignment α is an infinite bit string, whose first $N = m' \cdot H$ bits are supposed to encode the first m' symbols on M's input tape, which contains the pair $\langle x, u\rangle$. The first step of the proof is thus to extract a u of length $p(n)$ from the first N bits of α. The Formula Φ_7 ensures that the symbols representing u are **0** or **1** and thus represent a bit string.

Next the proof shows that the first N bits of α encode the relevant portion $y(u)$ of the input tape for the u just extracted, that is, $y(u)_i = \alpha(\gamma_i)$ for $i < m'$. The proof exploits the constraints set by Φ_1 to Φ_6. In particular this implies that $y(u)_{inputpos(t)} = \alpha(\gamma_{inputpos(t)})$ for all t.

The next $Z \cdot (T' + 1)$ bits of α encode $T' + 1$ snapshots. More precisely, we prove $z_0(u,t) = \alpha(\zeta_0^t)$ and $z_1(u,t) = \alpha(\zeta_1^t)$ and $z_2(u,t) = \alpha(\zeta_2^t)$ for all $t \leq T'$. This works by induction on t. The case $t = 0$ is covered by the formula Φ_0. For $0 < t \leq T'$ the formulas χ_t are responsible, which make up Φ_9. Basically χ_t represents the recursive characterization of the snapshot z_t in terms of earlier snapshots (of $t-1$ and possibly $prev(t)$). This is the trickiest part and we need some preliminary lemmas for that.

Once that is done, we know that some bits of α, namely $\alpha(\zeta_1(T'))$, encode the symbol under the output tape head after T' steps, that is, when M has halted. Formula Φ_8 ensures that this symbol is **1**, which concludes the proof.

lemma *sat-PHI-imp-ex-u*:
  assumes *satisfiable* Φ
  shows $\exists u.\ length\ u = p\ n \wedge exc\ \langle x;\ u\rangle\ T' <.> 1 = \mathbf{1}$
proof −
  obtain α **where** α: $\alpha \models \Phi$
    using *assms satisfiable-def* **by** *auto*

  define *us* **where** $us = map\ (\lambda i.\ unary\ \alpha\ (\gamma\ (2*n+4+2*i)))\ [0..<p\ n]$

  have *len-us*: $length\ us = p\ n$

```
  using us-def by simp

have us23: us ! i = 2 ∨ us ! i = 3 if i < p n for i
proof -
  have α ⊨ Φ7
    using PHI-def satisfies-def α by simp
  then have α ⊨ Υ (γ (2*n+4+2*i))
    using that PHI7-def satisfies-concat-map by auto
  then have unary α (γ (2*n+4+2*i)) = 2 ∨ unary α (γ (2*n+4+2*i)) = 3
    using Upsilon-unary length-gamma H-gr-2 by simp
  then show ?thesis
    using us-def that by simp
qed

define u :: string where u = symbols-to-string us

have len-u: length u = p n
  using u-def len-us by simp

have ysymbols u ! i = unary α (γ i) if i < m' for i
proof -
  consider
      i = 0
    | 1 ≤ i ∧ i < 2 * n + 1
    | 2 * n + 1 ≤ i ∧ i < 2 * n + 3
    | 2 * n + 3 ≤ i ∧ i < m + 1
    | i ≥ m + 1 ∧ i < m + 1 + T'
    using ‹i < m'› m'-def m-def by linarith
  then show ?thesis
  proof (cases)
    case 1
    then have α ⊨ Ψ (γ i) 1
      using PHI-def PHI1-def α satisfies-append by metis
    then have unary α (γ i) = 1
      using Psi-unary H-gr-2 gamma-def by simp
    moreover have ysymbols u ! i = 1
      using 1 by (simp add: ysymbols-def)
    ultimately show ?thesis
      by simp
  next
    case 2
    define j where j = (i - 1) div 2
    then have j < n
      using 2 by auto
    have i = 2 * j + 1 ∨ i = 2 * j + 2
      using 2 j-def by auto
    then show ?thesis
    proof
      assume i: i = 2 * j + 1
      have α ⊨ Φ3
        using PHI-def satisfies-def α by simp
      then have α ⊨ Ψ (γ (2*j+1)) 2
        using PHI3-def satisfies-concat-map[OF - ‹j < n›] by auto
      then have unary α (γ (2*j+1)) = 2
        using Psi-unary H-gr-2 gamma-def by simp
      moreover have ysymbols u ! (2*j+1) = 2
        using ysymbols-first-at[OF ‹j < n›] by simp
      ultimately show ?thesis
        using i by simp
    next
      assume i: i = 2 * j + 2
      have α ⊨ Φ6
        using PHI-def satisfies-def α by simp
```

```
      then have α ⊨ Ψ (γ (2*j+2)) (if x ! j then 3 else 2)
        using PHI6-def satisfies-concat-map[OF - ‹j < n›] by fastforce
      then have unary α (γ (2*j+2)) = (if x ! j then 3 else 2)
        using Psi-unary H-gr-2 gamma-def by simp
      moreover have ysymbols u ! (2*j+2) = (if x ! j then 3 else 2)
        using ysymbols-first-at[OF ‹j < n›] by simp
      ultimately show ?thesis
        using i by simp
    qed
  next
    case 3
    then have i = 2*n + 1 ∨ i = 2*n + 2
      by auto
    then show ?thesis
    proof
      assume i: i = 2 * n + 1
      then have α ⊨ Ψ (γ i) 3
        using PHI-def PHI2-def α satisfies-append by metis
      then have unary α (γ i) = 3
        using Psi-unary H-gr-2 gamma-def by simp
      moreover have ysymbols u ! i = 3
        using i ysymbols-at-2n1 by simp
      ultimately show ?thesis
        by simp
    next
      assume i: i = 2 * n + 2
      then have α ⊨ Ψ (γ i) 3
        using PHI-def PHI2-def α satisfies-append by metis
      then have unary α (γ i) = 3
        using Psi-unary H-gr-2 gamma-def by simp
      moreover have ysymbols u ! i = 3
        using i ysymbols-at-2n2 by simp
      ultimately show ?thesis
        by simp
    qed
  next
    case 4
    moreover define j where j = (i − 2*n−3) div 2
    ultimately have j: j < p n
      using m-def by auto
    have i = 2*n+2+2 * j + 1 ∨ i = 2*n+2+2 * j + 2
      using 4 j-def by auto
    then show ?thesis
    proof
      assume i: i = 2 * n + 2 + 2 * j + 1
      have α ⊨ Φ₄
        using PHI-def satisfies-def α by simp
      then have α ⊨ Ψ (γ (2*n+2+2*j+1)) 2
        using PHI4-def satisfies-concat-map[OF - ‹j < p n›] by auto
      then have unary α (γ (2*n+2+2*j+1)) = 2
        using Psi-unary H-gr-2 gamma-def by simp
      moreover have ysymbols u ! (2*n+2+2*j+1) = 2
        using ‹j < p n› ysymbols-second-at(1) len-u by presburger
      ultimately show ?thesis
        using i by simp
    next
      assume i: i = 2 * n + 2 + 2 * j + 2
      have us ! j = unary α (γ (2*n+4+2*j))
        using us-def ‹j < p n› by simp
      then have us ! j = unary α (γ (2*n+2+2*j+2))
        by (simp add: numeral-Bit0)
      then have unary α (γ (2*n+2+2*j+2)) = (if u ! j then 3 else 2)
        using u-def us23 ‹j < p local.n› len-us by fastforce
```

    then have ∗: *unary* α (γ *i*) = (*if u* ! *j then 3 else 2*)
      using *i* **by** *simp*
    have *ysymbols u* ! (*2∗n+2+2∗j+2*) = (*if u* ! *j then 3 else 2*)
      using *ysymbols-second-at*(*2*) ‹*j* < *p n*› *len-u* **by** *simp*
    then have *ysymbols u* ! *i* = (*if u* ! *j then 3 else 2*)
      using *i* **by** *simp*
    then show *?thesis*
      using ∗ *i* **by** *simp*
  qed
next
  case *5*
  then have $\alpha \models \Phi_5$
    using *PHI-def satisfies-def* α **by** *simp*
  then have $\alpha \models \Psi$ (γ (*2∗n+2∗p n* + *3* + *i*′)) *0* **if** *i*′ < *T*′ **for** *i*′
    unfolding *PHI5-def* **using** α *satisfies-concat-map*[*OF - that*, *of* α] *that* **by** *auto*
  moreover obtain *i*′ **where** *i*′: *i*′ < *T*′ *i* = *m* + *1* + *i*′
    using *5* **by** (*metis add-less-cancel-left le-Suc-ex*)
  ultimately have $\alpha \models \Psi$ (γ *i*) *0*
    using *m-def numeral-3-eq-3* **by** *simp*
  then have *unary* α (γ *i*) = *0*
    using *Psi-unary H-gr-2 gamma-def* **by** *simp*
  moreover have *ysymbols u* ! *i* = *0*
    using *5 ysymbols-last len-u* **by** *simp*
  ultimately show *?thesis*
    by *simp*
qed
qed
then have *ysymbols*: *ysymbols u* ! (*inputpos m t*) = *unary* α (γ (*inputpos m t*)) **for** *t*
  using *inputpos-less*′ *len-u* **by** *simp*

have *z0 u t* = *unary* α (ζ_0 *t*) ∧ *z1 u t* = *unary* α (ζ_1 *t*) ∧ *z2 u t* = *unary* α (ζ_2 *t*)
    if *t* ≤ *T*′ **for** *t*
  using *that*
proof (*induction t rule*: *nat-less-induct*)
  case (*1 t*)
  show *?case*
  proof (*cases t* = *0*)
    case *True*
    have $\alpha \models \Phi_0$
      using α *PHI-def satisfies-def* **by** *simp*
    then have
      1: $\alpha \models \Psi$ (ζ_0 *0*) *1* **and**
      2: $\alpha \models \Psi$ (ζ_1 *0*) *1* **and**
      3: $\alpha \models \Psi$ (ζ_2 *0*) *0*
      using *PHI0-def* **by** (*metis satisfies-append*(*1*), *metis satisfies-append*, *metis satisfies-append*(*2*))
    have *unary* α (ζ_0 *0*) = *1*
      using *Psi-unary*[*OF - 1*] *H-gr-2* **by** *simp*
    moreover have *unary* α (ζ_1 *0*) = *1*
      using *Psi-unary*[*OF - 2*] *H-gr-2* **by** *simp*
    moreover have *unary* α (ζ_2 *0*) = *0*
      using *Psi-unary*[*OF - 3*] **by** *simp*
    moreover have *z0 u 0* = ▷
      using *z0-def start-config2* **by** (*simp add*: *start-config-pos*)
    moreover have *z1 u 0* = ▷
      using *z1-def* **by** (*simp add*: *start-config2 start-config-pos*)
    moreover have *z2 u 0* = □
      using *z2-def* **by** (*simp add*: *start-config-def*)
    ultimately show *?thesis*
      using *True* **by** *simp*
  next
    case *False*
    then have *t* > *0*
      by *simp*

```
      let ?t = t - 1
      have sat-chi: α ⊨ χ t
      proof -
        have α ⊨ Φ₉
          using α PHI-def satisfies-def by simp
        moreover have ?t < T'
          using 1 ‹t > 0› by simp
        ultimately have α ⊨ χ (Suc ?t)
          using satisfies-concat-map PHI9-def by auto
        then show ?thesis
          using ‹t > 0› by simp
      qed
      show ?thesis
      proof (cases prev m t < t)
        case True
        then show ?thesis
          using satisfies-chi-less z0 z1 z2' 1 len-u ysymbols sat-chi True ‹t ≤ T'› len-u by simp
      next
        case False
        then have eq: prev m t = t
          using prev-eq prev-less by blast
        show ?thesis
          using satisfies-chi-eq z0 z1' z2' ysymbols sat-chi eq ‹t > 0› ‹t ≤ T'› len-u 1 ‹t > 0› by simp
      qed
    qed
  qed
  then have z1 u T' = unary α (ζ₁ T')
    by simp
  moreover have unary α (ζ₁ T') = 3
  proof -
    have α ⊨ Φ₈
      using α PHI-def satisfies-def by simp
    then have α ⊨ Ψ (ζ₁ T') 3
      using PHI8-def by simp
    then show ?thesis
      using Psi-unary[of 3 ζ₁ T'] length-zeta1 H-gr-2 by simp
  qed
  ultimately have z1 u T' = 1
    by simp
  then have exc ⟨x; u⟩ T' <.> 1 = 1
    using z1-def by simp
  then show ?thesis
    using len-u by auto
qed
```

6.6.2 $x \in L$ implies Φ satisfiable

For the other direction, we assume a string $x \in L$ and show that the formula Φ derived from it is satisfiable. From $x \in L$ it follows that there is a certificate u of length $p(n)$ such that M on input $\langle x, u\rangle$ halts after T' steps with the output tape head on the symbol **1**.

An assignment that satisfies Φ must have its first $N = m' \cdot H$ bits set in such a way that they encode the relevant portion $y(u)$ of the input tape, that is, with $\langle x, u\rangle$ followed by T' blanks. This will take care of satisfying $\Phi_1, \ldots, \Phi_7$. The next $Z \cdot (T' + 1)$ bits of α must be set such that they encode the $T' + 1$ snapshots of M when started on $\langle x, u\rangle$. This way Φ_0 and Φ_9 will be satisfied. Finally, Φ_8 is satisfied because by the choice of u the last snapshot contains a **1** as the symbol under the output tape head.

The following function maps a string u to an assignment as just described.

```
definition beta :: string ⇒ assignment (‹β›) where
  β u i ≡
    if i < N then
      let
        j = i div H;
        k = i mod H
```

```
    in
      if j = 0 then k < 1
      else if j = 2 * n + 1 ∨ j = 2 * n + 2 then k < 3
      else if j ≥ 2 * n + 2 * p n + 3 then k < 0
      else if odd j then k < 2
      else if j ≤ 2 * n then k < (if x ! (j div 2 − 1) then 3 else 2)
      else k < (if u ! (j div 2 − n − 2) then 3 else 2)
  else if i < N + Z * (Suc T') then
    let t = (i − N) div Z; k = (i − N) mod Z in
      if k < H then k < z0 u t
      else if k < 2 * H then k − H < z1 u t
      else k − 2 * H < z2 u t
  else False
```

In order to show that $\beta(u)$ satisfies Φ, we show that it satisfies all parts of Φ. These parts consist mostly of Ψ formulas, whose satisfiability can be proved using lemma *satisfies-Psi*. To apply this lemma, the following ones will be helpful.

```
lemma blocky-gammaI:
  assumes ⋀k. k < H ⟹ α (j * H + k) = (k < l)
  shows blocky α (γ j) l
  unfolding blocky-def gamma-def using assms by simp

lemma blocky-zeta0I:
  assumes ⋀k. k < H ⟹ α (N + t * Z + k) = (k < l)
  shows blocky α (ζ₀ t) l
  unfolding blocky-def zeta0-def using assms by simp

lemma blocky-zeta1I:
  assumes ⋀k. k < H ⟹ α (N + t * Z + H + k) = (k < l)
  shows blocky α (ζ₁ t) l
  unfolding blocky-def zeta1-def using assms by simp

lemma blocky-zeta2I:
  assumes ⋀k. k < H ⟹ α (N + t * Z + 2 * H + k) = (k < l)
  shows blocky α (ζ₂ t) l
  unfolding blocky-def zeta2-def using Z-def assms by simp

lemma beta-1: blocky (β u) (γ 0) 1
proof (intro blocky-gammaI)
  fix k :: nat
  assume k: k < H
  let ?i = 0 * H + k
  have ?i < N
    using N-eq add-mult-distrib2 k by auto
  then show β u ?i = (k < 1)
    using beta-def k by simp
qed

lemma beta-2a: blocky (β u) (γ (2*n+1)) 3
proof (intro blocky-gammaI)
  fix k :: nat
  assume k: k < H
  let ?i = (2*n+1) * H + k
  let ?j = ?i div H
  let ?k = ?i mod H
  have ?i < N
    using N-eq add-mult-distrib2 k by auto
  moreover have j: ?j = 2 * n + 1
    using k by (metis add-cancel-left-right div-less div-mult-self3 less-nat-zero-code)
  moreover have ?k = k
    using k by (metis mod-if mod-mult-self3)
  moreover have ¬ ?j = 0
    using j by linarith
```

ultimately show *β u ?i* = (*k* < *3*)
  using *beta-def* **by** *simp*
qed

lemma *beta-2b*: *blocky* (*β u*) (*γ* (*2*∗*n*+*2*)) *3*
proof (*intro blocky-gammaI*)
  fix *k* :: *nat*
  assume *k*: *k* < *H*
  let *?i* = (*2*∗*n*+*2*) ∗ *H* + *k*
  let *?j* = *?i div H*
  let *?k* = *?i mod H*
  have *?i* < *N*
    using *N-eq add-mult-distrib2 k* **by** *auto*
  moreover have *?j* = *2* ∗ *n* + *2*
    using *k* **by** (*metis add-cancel-left-right div-less div-mult-self3 less-nat-zero-code*)
  moreover have *?k* = *k*
    using *k* **by** (*metis mod-if mod-mult-self3*)
  moreover have ¬ *?j* = *0*
    using *calculation*(*2*) **by** *linarith*
  ultimately show *β u ?i* = (*k* < *3*)
    using *beta-def Let-def k* **by** *presburger*
qed

lemma *beta-3*:
  assumes *ii* < *n*
  shows *blocky* (*β u*) (*γ* (*2* ∗ *ii* + *1*)) *2*
proof (*intro blocky-gammaI*)
  fix *k* ::*nat*
  assume *k*: *k* < *H*
  let *?i* = (*2*∗*ii*+*1*) ∗ *H* + *k*
  let *?j* = *?i div H*
  let *?k* = *?i mod H*
  have *?i* < *N*
  proof −
    have *?i* < (*2*∗*n*+*1*) ∗ *H* + *k*
      using *assms k* **by** *simp*
    also have ... < (*2*∗*n*+*1*) ∗ *H* + *H*
      using *k* **by** *simp*
    also have ... = *H* ∗ (*2*∗*n*+*2*)
      by *simp*
    also have ... ≤ *H* ∗ (*2*∗*n*+*3*)
      by (*metis add.commute add.left-commute add-2-eq-Suc le-add2 mult-le-mono2 numeral-3-eq-3*)
    also have ... ≤ *H* ∗ (*2*∗*n*+*2*∗*p n*+*3*)
      by *simp*
    also have ... ≤ *H* ∗ (*2*∗*n*+*2*∗*p n*+*3* + *T′*)
      by *simp*
    finally have *?i* < *H* ∗ (*2*∗*n*+*2*∗*p n*+*3* + *T′*) **.**
    then show *?thesis*
      using *N-eq* **by** *simp*
  qed
  moreover have *j*: *?j* = *2* ∗ *ii* + *1*
    using *k* **by** (*metis add-cancel-left-right div-less div-mult-self3 less-nat-zero-code*)
  moreover have *?k* = *k*
    using *k* **by** (*metis mod-if mod-mult-self3*)
  moreover have ¬ *?j* = *0*
    using *j* **by** *linarith*
  moreover have ¬ (*?j* = *2*∗*n*+*1* ∨ *?j* = *2*∗*n*+*2*)
    using *j assms* **by** *simp*
  moreover have ¬ *?j* ≥ *2*∗*n*+*2*∗*p n* + *3*
    using *j assms* **by** *simp*
  moreover have *odd ?j*
    using *j* **by** *simp*
  ultimately show *β u ?i* = (*k* < *2*)

```
    using beta-def by simp
qed

lemma beta-4:
  assumes ii < p n and length u = p n
  shows blocky (β u) (γ (2*n+2+2*ii+1)) 2
proof (intro blocky-gammaI)
  fix k :: nat
  assume k: k < H
  let ?i = (2*n+2+2*ii+1) * H + k
  let ?j = ?i div H
  let ?k = ?i mod H
  have ?i < N
  proof -
    have ?i = (2*n+2*ii+3) * H + k
      by (simp add: numeral-3-eq-3)
    also have ... < (2*n+2*ii+3) * H + H
      using k by simp
    also have ... = H * (2*n+2*ii+4)
      by algebra
    also have ... ≤ H * (2*n+2*p n+3)
      using assms(1) by simp
    also have ... ≤ H * (2*n+2*p n+3 + T')
      by simp
    finally have ?i < H * (2*n+2*p n+3 + T') .
    then show ?thesis
      using N-eq by simp
  qed
  moreover have j: ?j = 2 * n + 2 + 2 * ii + 1
    using k by (metis add-cancel-left-right div-less div-mult-self3 less-nat-zero-code)
  moreover have ?k = k
    using k by (metis mod-if mod-mult-self3)
  moreover have ¬ ?j = 0
    using j by linarith
  moreover have ¬ (?j = 2*n+1 ∨ ?j = 2*n+2)
    using j assms by simp
  moreover have ¬ ?j ≥ 2*n+2*p n + 3
    using j assms by simp
  moreover have odd ?j
    using j by simp
  ultimately show β u ?i = (k < 2)
    using beta-def by simp
qed

lemma beta-5:
  assumes ii < T'
  shows blocky (β u) (γ (2*n+2*p n + 3 + ii)) 0
proof (intro blocky-gammaI)
  fix k :: nat
  assume k: k < H
  let ?i = (2*n+2*p n + 3 + ii) * H + k
  let ?j = ?i div H
  let ?k = ?i mod H
  have ?i < N
  proof -
    have ?i < (2*n+2*p n + 3 + ii) * H + H
      using k by simp
    also have ... ≤ (2*n+2*p n + 3 + T' − 1) * H + H
    proof -
      have 2*n+2*p n + 3 + ii ≤ 2*n+2*p n + 3 + T' − 1
        using assms by simp
      then show ?thesis
        using add-le-mono1 mult-le-mono1 by presburger
```

```
    qed
    also have ... ≤ (2*n+2*p n + 2 + T') * H + H
      by simp
    also have ... ≤ H * (2*n+2*p n + 3 + T')
      by (simp add: numeral-3-eq-3)
    finally have ?i < H * (2*n+2*p n + 3 + T') .
    then show ?thesis
      using N-eq by simp
  qed
  moreover have j: ?j = 2 * n + 2*p n + 3 + ii
    using k by (metis add-cancel-left-right div-less div-mult-self3 less-nat-zero-code)
  moreover have ?k = k
    using k by (metis mod-if mod-mult-self3)
  moreover have ¬ ?j = 0
    using j by linarith
  moreover have ¬ (?j = 2*n+1 ∨ ?j = 2*n+2)
    using j by simp
  ultimately show β u ?i = (k < 0)
    using beta-def Let-def k by simp
qed

lemma beta-6:
  assumes ii < n
  shows blocky (β u) (γ (2 * ii + 2)) (if x ! ii then 3 else 2)
proof (intro blocky-gammaI)
  fix k :: nat
  assume k: k < H
  let ?i = (2*ii+2) * H + k
  let ?j = ?i div H
  let ?k = ?i mod H
  have ?i < N
  proof -
    have ?i ≤ (2*n+2) * H + k
      using assms by simp
    also have ... < (2*n+2) * H + H
      using k by simp
    also have ... = (2*n+3) * H
      by algebra
    also have ... ≤ (2*n + 2*p n + 3) * H
      by simp
    also have ... ≤ (2*n + 2*p n + 3 + T') * H
      by simp
    finally have ?i < (2*n + 2*p n + 3 + T') * H .
    then show ?thesis
      using N-eq by (metis mult.commute)
  qed
  moreover have j: ?j = 2 * ii + 2
    using k by (metis add-cancel-left-right div-less div-mult-self3 less-nat-zero-code)
  moreover have ?k = k
    using k by (metis mod-if mod-mult-self3)
  moreover have ¬ ?j = 0
    using j by linarith
  moreover have ¬ (?j = 2*n+1 ∨ ?j = 2*n+2)
    using j assms by simp
  moreover have ¬ ?j = 2*n+2*p n + 3
    using j assms by simp
  moreover have ¬ odd ?j
    using j by simp
  moreover have ?j ≤ 2 * n
    using j assms by simp
  ultimately show β u ?i = (k < (if x ! ii then 3 else 2))
    using beta-def by simp
qed
```

```
lemma beta-7:
  assumes ii < p n and length u = p n
  shows blocky (β u) (γ (2 ∗ n + 4 + 2 ∗ ii)) (if u ! ii then 3 else 2)
proof (intro blocky-gammaI)
  fix k :: nat
  assume k: k < H
  let ?i = (2∗n+4+2∗ii) ∗ H + k
  let ?j = ?i div H
  let ?k = ?i mod H
  have ?i < N
  proof −
    have 1: ii ≤ p n − 1
      using assms(1) by simp
    have 2: p n > 0
      using assms(1) by simp
    have ?i ≤ (2∗n+4+2∗(p n − 1)) ∗ H + k
      using 1 by simp
    also have ... = (2∗n+4+2∗p n−2) ∗ H + k
      using 2 diff-mult-distrib2 by auto
    also have ... = (2∗n+2+2∗p n) ∗ H + k
      by simp
    also have ... < (2∗n+2+2∗p n) ∗ H + H
      using k by simp
    also have ... = (2∗n+3+2∗p n) ∗ H
      by algebra
    also have ... = H ∗ (2∗n + 2∗p n + 3)
      by simp
    also have ... ≤ H ∗ (2∗n + 2∗p n + 3 + T′)
      by simp
    finally have ?i < H ∗ (2∗n + 2∗p n + 3 + T′) .
    then show ?thesis
      using N-eq by simp
  qed
  moreover have j: ?j = 2 ∗ n + 4 + 2 ∗ ii
    using k by (metis add-cancel-left-right div-less div-mult-self3 less-nat-zero-code)
  moreover have ?k = k
    using k by (metis mod-if mod-mult-self3)
  moreover have ¬ ?j = 0
    using j by linarith
  moreover have ¬ (?j = 2∗n+1 ∨ ?j = 2∗n+2)
    using j assms by simp
  moreover have ¬ odd ?j
    using j by simp
  moreover have ¬ ?j ≤ 2 ∗ n
    using j assms by simp
  moreover have ?j ≤ 2 ∗ n + 2 ∗ p n + 2
    using j assms by simp
  ultimately show β u ?i = (k < (if u ! ii then 3 else 2))
    using beta-def by simp
qed

lemma beta-zeta0:
  assumes t ≤ T′
  shows blocky (β u) (ζ0 t) (z0 u t)
proof (intro blocky-zeta0I)
  fix k ::nat
  assume k: k < H
  let ?i = N + t ∗ Z + k
  let ?t = (?i − N) div Z
  let ?k = (?i − N) mod Z
  have ¬ ?i < N
    by simp
```

```
  moreover have ?i < N + Z * (Suc T')
  proof -
    have ?i ≤ N + T' * Z + k
      using assms by simp
    also have ... < N + T' * Z + H
      using k by simp
    also have ... ≤ N + T' * Z + Z
      using Z-def by simp
    also have ... = N + Z * (Suc T')
      by simp
    finally show ?thesis
      by simp
  qed
  moreover have kk: ?k = k
    using k Z-def by simp
  moreover have ?t = t
    using k Z-def by simp
  moreover have ?k < H
    using kk k by simp
  ultimately show β u ?i = (k < z0 u t)
    using beta-def by simp
qed

lemma beta-zeta1:
  assumes t ≤ T'
  shows blocky (β u) (ζ₁ t) (z1 u t)
proof (intro blocky-zeta1I)
  fix k :: nat
  assume k: k < H
  let ?i = N + t * Z + H + k
  let ?t = (?i - N) div Z
  let ?k = (?i - N) mod Z
  have ¬ ?i < N
    by simp
  moreover have ?i < N + Z * (Suc T')
  proof -
    have ?i ≤ N + T' * Z + H + k
      using assms by simp
    also have ... < N + T' * Z + H + H
      using k by simp
    also have ... ≤ N + T' * Z + Z
      using Z-def by simp
    also have ... = N + Z * (Suc T')
      by simp
    finally show ?thesis
      by simp
  qed
  moreover have ?t = t
    using k Z-def by simp
  moreover have kk: ?k = H + k
    using k Z-def by simp
  moreover have ¬ ?k < H
    using kk by simp
  moreover have ?k < 2 * H
    using kk k by simp
  ultimately have β u ?i = (?k - H < z1 u t)
    using beta-def by simp
  then show β u ?i = (k < z1 u t)
    using kk by simp
qed

lemma beta-zeta2:
  assumes t ≤ T'
```

shows *blocky* (β *u*) (ζ_2 *t*) (*z2 u t*)
proof (*intro blocky-zeta2I*)
  fix *k* :: *nat*
  assume *k*: $k < H$
  let $?i = N + t * Z + 2 * H + k$
  let $?t = (?i - N)$ *div* Z
  let $?k = (?i - N)$ *mod* Z
  have *1*: $2 * H + k < Z$
    using *k Z-def* **by** *simp*
  have $\neg\ ?i < N$
    by *simp*
  moreover have $?i < N + Z * (Suc\ T')$
  proof −
    have $?i \leq N + T' * Z + 2 * H + k$
      using *assms* **by** *simp*
    also have ... $< N + T' * Z + 2 * H + H$
      using *k* **by** *simp*
    also have ... $\leq N + T' * Z + Z$
      using *Z-def* **by** *simp*
    also have ... $= N + Z * (Suc\ T')$
      by *simp*
    finally show *?thesis*
      by *simp*
  qed
  moreover have $?t = t$
    using *1* **by** *simp*
  moreover have *kk*: $?k = 2 * H + k$
    using *k Z-def* **by** *simp*
  moreover have $\neg\ ?k < H$
    using *kk* **by** *simp*
  moreover have $\neg\ ?k < 2 * H$
    using *kk* **by** *simp*
  ultimately have $\beta\ u\ ?i = (?k - 2 * H < z2\ u\ t)$
    using *beta-def* **by** *simp*
  then show $\beta\ u\ ?i = (k < z2\ u\ t)$
    using *kk* **by** *simp*
qed

We can finally show that $\beta(u)$ satisfies Φ if u is a certificate for x.

lemma *satisfies-beta-PHI*:
  assumes *length* $u = p\ n$ **and** *exc* $\langle x; u\rangle\ T'$ <.> $1 = \mathbf{1}$
  shows $\beta\ u \models \Phi$
proof −
  have $\beta\ u \models \Phi_0$
  proof −
    have *blocky* (β *u*) (ζ_0 *0*) (*z0 u 0*)
      using *beta-zeta0* **by** *simp*
    then have *blocky* (β *u*) (ζ_0 *0*) *1*
      using *z0-def start-config2 start-config-pos* **by** *auto*
    then have $\beta\ u \models \Psi\ (\zeta_0\ 0)\ 1$
      using *satisfies-Psi H-gr-2* **by** *simp*
    moreover have $\beta\ u \models \Psi\ (\zeta_1\ 0)\ 1$
    proof −
      have *blocky* (β *u*) (ζ_1 *0*) (*z1 u 0*)
        using *beta-zeta1* **by** *simp*
      then have *blocky* (β *u*) (ζ_1 *0*) *1*
        using *z1-def start-config2 start-config-pos* **by** *simp*
      then show *?thesis*
        using *satisfies-Psi H-gr-2* **by** *simp*
    qed
    moreover have $\beta\ u \models \Psi\ (\zeta_2\ 0)\ 0$
    proof −
      have *blocky* (β *u*) (ζ_2 *0*) (*z2 u 0*)

using *beta-zeta2* **by** *simp*
then have *blocky* (β *u*) (ζ_2 *0*) *0*
using *z2-def start-config-def* **by** *simp*
then show *?thesis*
using *satisfies-Psi H-gr-2* **by** *simp*
qed
ultimately show *?thesis*
using *PHI0-def satisfies-def* **by** *auto*
qed
moreover have β *u* $\models \Phi_1$
using *PHI1-def H-gr-2 satisfies-Psi beta-1* **by** *simp*
moreover have β *u* $\models \Phi_2$
proof −
have β *u* $\models \Psi$ (γ (*2*∗*n*+*1*)) *3*
using *satisfies-Psi H-gr-2 beta-2a* **by** *simp*
moreover have β *u* $\models \Psi$ (γ (*2*∗*n*+*2*)) *3*
using *satisfies-Psi H-gr-2 beta-2b* **by** *simp*
ultimately show *?thesis*
using *PHI2-def satisfies-def* **by** *auto*
qed
moreover have β *u* $\models \Phi_3$
proof −
have β *u* $\models \Psi$ (γ (*2*∗*i*+*1*)) *2* **if** *i* < *n* **for** *i*
using *satisfies-Psi that H-gr-2 length-gamma less-imp-le-nat beta-3* **by** *simp*
then show *?thesis*
using *PHI3-def satisfies-concat-map′* **by** *simp*
qed
moreover have β *u* $\models \Phi_4$
proof −
have β *u* $\models \Psi$ (γ (*2*∗*n*+*2*+*2*∗*i*+*1*)) *2* **if** *i* < *p n* **for** *i*
using *satisfies-Psi that H-gr-2 length-gamma less-imp-le-nat beta-4 assms*(*1*) **by** *simp*
then show *?thesis*
using *PHI4-def satisfies-concat-map′* **by** *simp*
qed
moreover have β *u* $\models \Phi_5$
proof −
have β *u* $\models \Psi$ (γ (*2*∗*n*+*2*∗*p n*+*3*+*i*)) *0* **if** *i* < *T′* **for** *i*
using *satisfies-Psi that H-gr-2 length-gamma less-imp-le-nat beta-5 assms*(*1*) **by** *simp*
then show *?thesis*
using *PHI5-def satisfies-concat-map′* **by** *simp*
qed
moreover have β *u* $\models \Phi_6$
proof −
have β *u* $\models \Psi$ (γ (*2*∗*i*+*2*)) (*if x* ! *i then 3 else 2*) **if** *i* < *n* **for** *i*
using *satisfies-Psi that H-gr-2 length-gamma less-imp-le-nat beta-6* **by** *simp*
then show *?thesis*
using *PHI6-def satisfies-concat-map′* **by** *simp*
qed
moreover have β *u* $\models \Phi_7$
proof −
have β *u* $\models \Upsilon$ (γ (*2*∗*n*+*4*+*2*∗*i*)) **if** *i* < *p n* **for** *i*
proof −
have *blocky* (β *u*) (γ (*2*∗*n*+*4*+*2*∗*i*)) *2* ∨ *blocky* (β *u*) (γ (*2*∗*n*+*4*+*2*∗*i*)) *3*
using *assms that beta-7*[*of i u*] **by** (*metis* (*full-types*))
then have β *u* $\models \Psi$ (γ (*2*∗*n*+*4*+*2*∗*i*)) *2* ∨ β *u* $\models \Psi$ (γ (*2*∗*n*+*4*+*2*∗*i*)) *3*
using *satisfies-Psi H-gr-2* **by** *auto*
then show *?thesis*
using *Psi-2-imp-Upsilon Psi-3-imp-Upsilon H-gr-2 length-gamma* **by** *auto*
qed
then show *?thesis*
using *PHI7-def satisfies-concat-map′* **by** *simp*
qed
moreover have β *u* $\models \Phi_8$

using *PHI8-def H-gr-2 assms satisfies-Psi z1-def beta-zeta1*
by (*metis One-nat-def Suc-1 Suc-leI length-zeta1 nat-le-linear numeral-3-eq-3*)
moreover have $\beta\ u \models \Phi_9$
proof −
have ∗: *unary* (β *u*) (γ *i*) = *ysymbols u* ! *i* **if** *i* < *length* (*ysymbols u*) **for** *i*
proof −
have $i < m'$
using *assms length-ysymbols that* **by** *simp*
then consider
$i = 0$
| $1 \leq i \wedge i < 2*n + 1$
| $2*n+1 \leq i \wedge i < 2*n+3$
| $2*n+3 \leq i \wedge i < m+1$
| $i \geq m + 1 \wedge i < m + 1 + T'$
using *m′-def m-def* **by** *linarith*
then show *?thesis*
proof (*cases*)
case *1*
then show *?thesis*
using *ysymbols-at-0 blocky-imp-unary H-gr-2 beta-1* **by** *simp*
next
case *2*
moreover define *j* **where** $j = (i - 1)$ *div 2*
ultimately have *j*: $j < n$ $i = 2 * j + 1 \vee i = 2 * j + 2$
by *auto*
show *?thesis*
proof (*cases* $i = 2 * j + 1$)
case *True*
then show *?thesis*
using *ysymbols-first-at*(*1*) *blocky-imp-unary H-gr-2 j*(*1*) *beta-3* **by** *simp*
next
case *False*
then have $i = 2 * j + 2$
using *j*(*2*) **by** *simp*
then show *?thesis*
using *ysymbols-first-at*(*2*) *blocky-imp-unary H-gr-2 j*(*1*) *beta-4 beta-6* **by** *simp*
qed
next
case *3*
show *?thesis*
proof (*cases* $i = 2*n+1$)
case *True*
then show *?thesis*
using *ysymbols-at-2n1 blocky-imp-unary H-gr-2 beta-2a* **by** *simp*
next
case *False*
then have $i = 2*n+2$
using *3* **by** *simp*
then show *?thesis*
using *ysymbols-at-2n2 blocky-imp-unary H-gr-2 beta-2b* **by** *simp*
qed
next
case *4*
moreover define *j* **where** $j = (i - 2*n-3)$ *div 2*
ultimately have *j*: $j < p\ n$ $i = 2*n+2+2 * j + 1 \vee i = 2*n+2+2 * j + 2$
using *j-def m-def* **by** *auto*
show *?thesis*
proof (*cases* $i = 2*n+2+2 * j + 1$)
case *True*
then show *?thesis*
using *ysymbols-second-at*(*1*) *assms*(*1*) *blocky-imp-unary H-gr-2 j*(*1*) *beta-4* **by** *simp*
next
case *False*

then have *i*: $i = 2{*}n{+}4{+}2 * j$
using $j(2)$ **by** *simp*
then have *ysymbols u* ! $(2{*}n{+}2{+}2{*}j{+}2)$ = (*if u* ! *j then 3 else 2*)
using *ysymbols-second-at*(2) *assms j*(1) **by** *simp*
then have *ysymbols u* ! $(2{*}n{+}4{+}2{*}j)$ = (*if u* ! *j then 3 else 2*)
by (*metis False i j*(2))
then have *ysymbols u* ! *i* = (*if u* ! *j then 3 else 2*)
using *i* **by** *simp*
then show *?thesis*
using *beta-7*[*OF j*(1)] *blocky-imp-unary H-gr-2 length-gamma i assms*(1) **by** *simp*
qed
next
case *5*
then obtain *ii* **where** *ii*: $ii < T'$ $i = m + 1 + ii$
by (*metis le-iff-add nat-add-left-cancel-less*)
have *blocky* (β *u*) (γ $(2{*}n{+}2{*}p\ n + 3 + ii)$) *0*
using *beta-5*[*OF ii*(1)] **by** *simp*
then have *blocky* (β *u*) (γ *i*) *0*
using *ii*(2) *m-def numeral-3-eq-3* **by** *simp*
then have *unary* (β *u*) (γ *i*) = *0*
using *blocky-imp-unary* **by** *simp*
moreover have *ysymbols u* ! *i* = *0*
using *ysymbols-last*[*OF assms*(1)] *5* **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed
have β *u* $\models \chi$ (*Suc t*) (**is** β *u* $\models \chi$ *?t*)
if $t < T'$ **for** *t*
proof (*cases prev m ?t* < *?t*)
case *True*
have *t*: $?t \le T'$
using *that* **by** *simp*
then have *unary* (β *u*) (ζ_0 *?t*) = *z0 u ?t*
using *blocky-imp-unary z0-le beta-zeta0* **by** *simp*
moreover have *ysymbols u* ! (*inputpos m ?t*) = *unary* (β *u*) (γ (*inputpos m ?t*))
using $*$ *assms*(1) *inputpos-less'* *length-ysymbols* **by** *simp*
ultimately have *unary* (β *u*) (ζ_0 *?t*) = *unary* (β *u*) (γ (*inputpos m ?t*))
using *assms*(1) *z0* **by** *simp*
moreover have *unary* (β *u*) (ζ_1 *?t*) = *z1 u ?t*
using *beta-zeta1 blocky-imp-unary z1-le t* **by** *simp*
moreover have *unary* (β *u*) (ζ_2 *?t*) = *z2 u ?t*
using *beta-zeta2 blocky-imp-unary z2-le t* **by** *simp*
moreover have *unary* (β *u*) (ζ_0 (*prev m ?t*)) = *z0 u* (*prev m ?t*)
using *beta-zeta0 blocky-imp-unary z0-le t True* **by** *simp*
moreover have *unary* (β *u*) (ζ_1 (*prev m ?t*)) = *z1 u* (*prev m ?t*)
using *beta-zeta1 blocky-imp-unary z1-le t True* **by** *simp*
moreover have *unary* (β *u*) (ζ_2 (*prev m ?t*)) = *z2 u* (*prev m ?t*)
using *beta-zeta2 blocky-imp-unary z2-le t True* **by** *simp*
moreover have *unary* (β *u*) (ζ_0 (*?t* − *1*)) = *z0 u* (*?t* − *1*)
using *beta-zeta0 blocky-imp-unary z0-le t True* **by** *simp*
moreover have *unary* (β *u*) (ζ_1 (*?t* − *1*)) = *z1 u* (*?t* − *1*)
using *beta-zeta1 blocky-imp-unary z1-le t True* **by** *simp*
moreover have *unary* (β *u*) (ζ_2 (*?t* − *1*)) = *z2 u* (*?t* − *1*)
using *beta-zeta2 blocky-imp-unary z2-le t True* **by** *simp*
ultimately show *?thesis*
using *True assms*(1) *satisfies-chi-less*[*OF True*] *t z1 z2'*
by (*metis bot-nat-0.extremum less-nat-zero-code nat-less-le*)
next
case *False*
then have *prev*: *prev m ?t* = *?t*
using *prev-le* **by** (*meson le-neq-implies-less*)
have *t*: $?t \le T'$

using *that* **by** *simp*
then have *unary* (β *u*) (ζ_0 *?t*) = *z0 u ?t*
using *beta-zeta0 blocky-imp-unary z0-le* **by** *simp*
moreover have *ysymbols u* ! (*inputpos m ?t*) = *unary* (β *u*) (γ (*inputpos m ?t*))
using $*$ *assms*(*1*) *inputpos-less′ length-ysymbols* **by** *simp*
ultimately have *unary* (β *u*) (ζ_0 *?t*) = *unary* (β *u*) (γ (*inputpos m ?t*))
using *assms*(*1*) *z0* **by** *simp*
moreover have *unary* (β *u*) (ζ_1 *?t*) = *z1 u ?t*
using *beta-zeta1 blocky-imp-unary z1-le t* **by** *simp*
moreover have *z1 u ?t* = $\Box$
using *z1′ beta-zeta1 assms*(*1*) *prev t* **by** *simp*
moreover have *unary* (β *u*) (ζ_2 *?t*) = *z2 u ?t*
using *beta-zeta2 blocky-imp-unary z2-le t* **by** *simp*
moreover have *unary* (β *u*) (ζ_0 (*?t* − *1*)) = *z0 u* (*?t* − *1*)
using *beta-zeta0 blocky-imp-unary z0-le t* **by** *simp*
moreover have *unary* (β *u*) (ζ_1 (*?t* − *1*)) = *z1 u* (*?t* − *1*)
using *beta-zeta1 blocky-imp-unary z1-le t* **by** *simp*
moreover have *unary* (β *u*) (ζ_2 (*?t* − *1*)) = *z2 u* (*?t* − *1*)
using *beta-zeta2 blocky-imp-unary z2-le t* **by** *simp*
ultimately show *?thesis*
using *satisfies-chi-eq*[*OF prev*] *start-config2 start-config-pos t that z1-def z2 assms*(*1*)
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-less-eq add-diff-inverse-nat execute.simps*(*1*) *less-one n-not-Suc-n plus-1-eq-Suc*)
qed
then show *?thesis*
using *PHI9-def satisfies-concat-map′* **by** *presburger*
qed
ultimately show *?thesis*
using *satisfies-append′ PHI-def* **by** *simp*
qed

corollary *ex-u-imp-sat-PHI*:
assumes *length u* = *p n* **and** *exc* ⟨*x*; *u*⟩ *T′* <.> *1* = **1**
shows *satisfiable* Φ
using *satisfies-beta-PHI assms satisfiable-def* **by** *auto*

The formula Φ has the desired property:

theorem *L-iff-satisfiable*: $x \in L \longleftrightarrow$ *satisfiable* Φ
using *L-iff-ex-u ex-u-imp-sat-PHI sat-PHI-imp-ex-u* **by** *auto*

end

end

Chapter 7

Auxiliary Turing machines for reducing $\mathcal{NP}$ languages to SAT

theory *Aux-TM-Reducing*
  imports *Reducing*
begin

In the previous chapter we have seen how to reduce a language $L \in \mathcal{NP}$ to SAT by constructing for every string x a CNF formula Φ that is satisfiable iff. $x \in L$. To complete the Cook-Levin theorem it remains to show that there is a polynomial-time Turing machine that on input x outputs Φ. Constructing such a TM will be the subject of the rest of this article and conclude in the next chapter. This chapter introduces several TMs used in the construction.

7.1 Generating literals

Our representation of CNF formulas as lists of lists of numbers is based on a representation of literals as numbers. Our function *literal-n* encodes the positive literal v_i as the number $2i + 1$ and the negative literal $\bar{v}_i$ as $2i$. We already have the Turing machine *tm-times2* to cover the second case. Now we build a TM for the first case, that is, for doubling and incrementing.

definition *tm-times2incr* :: *tapeidx* $\Rightarrow$ *machine* **where**
  tm-times2incr j $\equiv$ *tm-times2 j* ;; *tm-incr j*

lemma *tm-times2incr-tm*:
  assumes $0 < j$ **and** $j < k$ **and** $G \geq 4$
  shows *turing-machine k G* (*tm-times2incr j*)
  unfolding *tm-times2incr-def* **using** *tm-times2-tm tm-incr-tm assms* **by** *simp*

lemma *transforms-tm-times2incrI* [*transforms-intros*]:
  fixes *j* :: *tapeidx*
  fixes *k* :: *nat* **and** *tps tps'* :: *tape list*
  assumes $k \geq 2$ **and** $j > 0$ **and** $j < k$ **and** *length tps* = *k*
  assumes *tps* ! *j* = ($\lfloor n \rfloor_N$, *1*)
  assumes *t* = *12* + *4* ∗ *nlength n*
  assumes *tps'* = *tps*[*j* := ($\lfloor$*Suc* (*2* ∗ *n*)$\rfloor_N$, *1*)]
  shows *transforms* (*tm-times2incr j*) *tps t tps'*
proof −
  define *tt* **where** *tt* = *10* + (*2* ∗ *nlength n* + *2* ∗ *nlength* (*2* ∗ *n*))
  have *transforms* (*tm-times2incr j*) *tps tt tps'*
    unfolding *tm-times2incr-def* **by** (*tform tps*: *tt-def assms*)
  moreover have *tt* ≤ *t*
  proof −
    have *tt* = *10* + *2* ∗ *nlength n* + *2* ∗ *nlength* (*2* ∗ *n*)
      using *tt-def* **by** *simp*
    also have ... ≤ *10* + *2* ∗ *nlength n* + *2* ∗ (*Suc* (*nlength n*))

proof −
have *nlength* (*2* ∗ *n*) ≤ *Suc* (*nlength n*)
by (*metis eq-imp-le gr0I le-SucI nat-0-less-mult-iff nlength-even-le*)
then show *?thesis*
by *simp*
qed
also have ... = *12* + *4* ∗ *nlength n*
by *simp*
finally show *?thesis*
using *assms*(*6*) **by** *simp*
qed
ultimately show *?thesis*
using *transforms-monotone* **by** *simp*
qed

lemma *literal-n-rename*:
assumes *v div 2* < *length* σ
shows *2* ∗ σ ! (*v div 2*) + *v mod 2* = (*literal-n* ∘ *rename* σ) (*n-literal v*)
proof (*cases even v*)
case *True*
then show *?thesis*
using *n-literal-def assms* **by** *simp*
next
case *False*
then show *?thesis*
using *n-literal-def assms* **by** *simp presburger*
qed

Combining *tm-times2* and *tm-times2incr*, the next Turing machine accepts a variable index i on tape j_1 and a flag b on tape j_2 and outputs on tape j_1 the encoding of the positive literal v_i or the negative literal $\bar{v}_i$ if b is positive or zero, respectively.

definition *tm-to-literal* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-to-literal j1 j2 ≡
IF λ*rs. rs* ! *j2* = □ *THEN*
tm-times2 j1
ELSE
tm-times2incr j1
ENDIF

lemma *tm-to-literal-tm*:
assumes *k* ≥ *2* **and** *G* ≥ *4* **and** *0* < *j1* **and** *j1* < *k* **and** *j2* < *k*
shows *turing-machine k G* (*tm-to-literal j1 j2*)
unfolding *tm-to-literal-def*
using *assms tm-times2-tm tm-times2incr-tm turing-machine-branch-turing-machine*
by *simp*

lemma *transforms-tm-to-literalI* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx*
fixes *tps tps′* :: *tape list* **and** *ttt k i b* :: *nat*
assumes *0* < *j1 j1* < *k j2* < *k 2* ≤ *k length tps* = *k*
assumes
tps ! *j1* = (⌊*i*⌋$_N$, *1*)
tps ! *j2* = (⌊*b*⌋$_N$, *1*)
assumes *ttt* = *13* + *4* ∗ *nlength i*
assumes *tps′* = *tps*
[*j1* := (⌊*2* ∗ *i* + (*if b* = *0 then 0 else 1*)⌋$_N$, *1*)]
shows *transforms* (*tm-to-literal j1 j2*) *tps ttt tps′*
unfolding *tm-to-literal-def*
proof (*tform tps*: *assms read-ncontents-eq-0*)
show *5* + *2* ∗ *nlength i* + *2* ≤ *ttt* **and** *12* + *4* ∗ *nlength i* + *1* ≤ *ttt*
using *assms*(*8*) **by** *simp-all*
qed

7.2 A Turing machine for relabeling formulas

In order to construct Φ_9, we must construct CNF formulas χ_t, which have the form $\varrho(\psi)$ or $\varrho'(\psi')$. So we need a Turing machine for relabeling formulas. In this section we devise a Turing machine that gets a substitution σ and a CNF formula φ and outputs $\sigma(\varphi)$. In order to bound its running time we first prove some bounds on the length of relabeled formulas.

7.2.1 The length of relabeled formulas

First we bound the length of the representation of a single relabeled clause. In the following lemma the assumption ensures that the substitution σ has a value for every variable in the clause.

```
lemma nllength-rename:
  assumes ∀ v∈set clause. v div 2 < length σ
  shows nllength (map (literal-n ∘ rename σ) (n-clause clause)) ≤ length clause ∗ Suc (nllength σ)
proof (cases σ = [])
  case True
  then show ?thesis
    using assms n-clause-def by simp
next
  case False
  let ?f = literal-n ∘ rename σ ∘ n-literal
  have ∗: map (literal-n ∘ rename σ) (n-clause clause) = map ?f clause
    using n-clause-def by simp

  have nlength (2 ∗ n + 1) ≤ Suc (nlength n) for n
    using nlength-times2plus1 by simp
  then have nlength (2 ∗ Max (set σ) + 1) ≤ Suc (nlength (Max (set σ)))
    by simp
  moreover have nlength (Max (set σ)) ≤ nllength σ − 1
    using False member-le-nllength-1 by simp
  ultimately have nlength (2 ∗ Max (set σ) + 1) ≤ Suc (nllength σ − 1)
    by simp
  then have ∗∗: nlength (2 ∗ Max (set σ) + 1) ≤ nllength σ
    using nllength-gr-0 False by simp

  have ?f n ≤ 2 ∗ (σ ! (n div 2)) + 1 if n div 2 < length σ for n
    using n-literal-def by (cases even n) simp-all
  then have ?f v ≤ 2 ∗ (σ ! (v div 2)) + 1 if v ∈ set clause for v
    using assms that by simp
  moreover have σ ! (v div 2) ≤ Max (set σ) if v ∈ set clause for v
    using that assms by simp
  ultimately have ?f v ≤ 2 ∗ Max (set σ) + 1 if v ∈ set clause for v
    using that by fastforce
  then have n ≤ 2 ∗ Max (set σ) + 1 if n ∈ set (map ?f clause) for n
    using that by auto
  then have nllength (map ?f clause) ≤ Suc (nlength (2 ∗ Max (set σ) + 1)) ∗ length (map ?f clause)
    using nllength-le-len-mult-max by blast
  also have ... = Suc (nlength (2 ∗ Max (set σ) + 1)) ∗ length clause
    by simp
  also have ... ≤ Suc (nllength σ) ∗ length clause
    using ∗∗ by simp
  finally have nllength (map ?f clause) ≤ Suc (nllength σ) ∗ length clause .
  then show ?thesis
    using ∗ by (metis mult.commute)
qed
```

Our upper bound for the length of the symbol representation of a relabeled formula is fairly crude. It is basically the length of the string resulting from replacing every symbol of the original formula by the entire substitution.

```
lemma nlllength-relabel:
  assumes ∀ clause∈set φ. ∀ v∈set (clause-n clause). v div 2 < length σ
  shows nlllength (formula-n (relabel σ φ)) ≤ Suc (nllength σ) ∗ nlllength (formula-n φ)
```

using *assms*
proof (*induction* φ)
case *Nil*
then show *?case*
by (*simp add*: *relabel-def*)
next
case (*Cons clause* φ)
let *?nclause* = *clause-n clause*
have $\forall$ *v*$\in$*set ?nclause*. *v div 2* < *length* σ
using *Cons.prems* **by** *simp*
then have *nllength* (*map* (*literal-n* $\circ$ *rename* σ) (*n-clause ?nclause*)) $\leq$ *length ?nclause* $*$ *Suc* (*nllength* σ)
using *nllength-rename* **by** *simp*
then have *nllength* (*map* (*literal-n* $\circ$ *rename* σ) *clause*) $\leq$ *length clause* $*$ *Suc* (*nllength* σ)
using *clause-n-def n-clause-n* **by** *simp*
moreover have *map* (*literal-n* $\circ$ *rename* σ) *clause* = *clause-n* (*map* (*rename* σ) *clause*)
using *clause-n-def* **by** *simp*
ultimately have $*$: *nllength* (*clause-n* (*map* (*rename* σ) *clause*)) $\leq$ *length clause* $*$ *Suc* (*nllength* σ)
by *simp*

have *formula-n* (*relabel* σ (*clause* # φ)) = *clause-n* (*map* (*rename* σ) *clause*) # *formula-n* (*relabel* σ φ)
by (*simp add*: *formula-n-def relabel-def*)
then have *nlllength* (*formula-n* (*relabel* σ (*clause* # φ))) =
nllength (*clause-n* (*map* (*rename* σ) *clause*)) + *1* + *nlllength* (*formula-n* (*relabel* σ φ))
using *nlllength-Cons* **by** *simp*
also have ... $\leq$ *length clause* $*$ *Suc* (*nllength* σ) + *1* + *nlllength* (*formula-n* (*relabel* σ φ))
using $*$ **by** *simp*
also have ... $\leq$ *length clause* $*$ *Suc* (*nllength* σ) + *1* + *Suc* (*nllength* σ) $*$ *nlllength* (*formula-n* φ)
using *Cons* **by** (*metis add-mono-thms-linordered-semiring*(*2*) *insert-iff list.set*(*2*))
also have ... = *1* + *Suc* (*nllength* σ) $*$ (*length clause* + *nlllength* (*formula-n* φ))
by *algebra*
also have ... $\leq$ *Suc* (*nllength* σ) $*$ (*1* + *length clause* + *nlllength* (*formula-n* φ))
by *simp*
also have ... $\leq$ *Suc* (*nllength* σ) $*$ (*1* + *nllength* (*clause-n clause*) + *nlllength* (*formula-n* φ))
using *length-le-nllength n-clause-def n-clause-n*
by (*metis add-Suc-shift add-le-cancel-right length-map mult-le-mono2 plus-1-eq-Suc*)
also have ... = *Suc* (*nllength* σ) $*$ (*nlllength* (*formula-n* (*clause* # φ)))
using *formula-n-def nlllength-Cons* **by** *simp*
finally show *?case* **.**
qed

A simple sufficient condition for the assumption in the previous lemma.

lemma *variables-σ*:
assumes *variables* $\varphi \subseteq$ {..<*length* σ}
shows $\forall$ *clause*$\in$*set* φ.$\forall$ *v*$\in$*set* (*clause-n clause*). *v div 2* < *length* σ
proof *standard+*
fix *clause* :: *clause* **and** *v* :: *nat*
assume *clause*: *clause* $\in$ *set* φ **and** *v*: *v* $\in$ *set* (*clause-n clause*)

obtain *i* **where** *i*: *i* < *length clause v* = *literal-n* (*clause* ! *i*)
using *v clause-n-def* **by** (*metis in-set-conv-nth length-map nth-map*)
then have *clause-i*: *clause* ! *i* = *n-literal v*
using *n-literal-n* **by** *simp*

show *v div 2* < *length* σ
proof (*cases even v*)
case *True*
then have *clause* ! *i* = *Neg* (*v div 2*)
using *clause-i n-literal-def* **by** *simp*
then have $\exists$ *c*$\in$*set* φ. *Neg* (*v div 2*) $\in$ *set c*
using *clause i*(*1*) **by** (*metis nth-mem*)
then have *v div 2* $\in$ *variables* φ
using *variables-def* **by** *auto*
then show *?thesis*

```
      using assms by auto
  next
    case False
    then have clause ! i = Pos (v div 2)
      using clause-i n-literal-def by simp
    then have ∃ c∈set φ. Pos (v div 2) ∈ set c
      using clause i(1) by (metis nth-mem)
    then have v div 2 ∈ variables φ
      using variables-def by auto
    then show ?thesis
      using assms by auto
  qed
qed
```

Combining the previous two lemmas yields this upper bound:

```
lemma nlllength-relabel-variables:
  assumes variables φ ⊆ {..<length σ}
  shows nlllength (formula-n (relabel σ φ)) ≤ Suc (nllength σ) * nlllength (formula-n φ)
  using assms variables-σ nlllength-relabel by blast
```

7.2.2 Relabeling clauses

Relabeling a CNF formula is accomplished by relabeling each of its clauses. In this section we devise a Turing machine for relabeling clauses. The TM accepts on tape j a list of numbers representing a substitution σ and on tape $j+1$ a clause represented as a list of numbers. It outputs on tape $j+2$ the relabeled clause, consuming the original clause on tape $j+1$ in the process.

```
definition tm-relabel-clause :: tapeidx ⇒ machine where
  tm-relabel-clause j ≡
    WHILE [] ; λrs. rs ! (j + 1) ≠ □ DO
      tm-nextract | (j + 1) (j + 3) ;;
      tm-mod2 (j + 3) (j + 4) ;;
      tm-div2 (j + 3) ;;
      tm-nth-inplace j (j + 3) | ;;
      tm-to-literal (j + 3) (j + 4) ;;
      tm-append (j + 2) (j + 3) ;;
      tm-setn (j + 3) 0 ;;
      tm-setn (j + 4) 0
    DONE ;;
    tm-cr (j + 2) ;;
    tm-erase-cr (j + 1)

lemma tm-relabel-clause-tm:
  assumes G ≥ 5 and j + 4 < k and 0 < j
  shows turing-machine k G (tm-relabel-clause j)
  unfolding tm-relabel-clause-def
 using assms tm-nextract-tm tm-mod2-tm tm-div2-tm tm-nth-inplace-tm tm-to-literal-tm tm-append-tm tm-setn-tm
    tm-cr-tm tm-erase-cr-tm Nil-tm turing-machine-loop-turing-machine
  by simp

locale turing-machine-relabel-clause =
  fixes j :: tapeidx
begin

definition tm1 ≡ tm-nextract | (j + 1) (j + 3)
definition tm2 ≡ tm1 ;; tm-mod2 (j + 3) (j + 4)
definition tm3 ≡ tm2 ;; tm-div2 (j + 3)
definition tm4 ≡ tm3 ;; tm-nth-inplace j (j + 3) |
definition tm5 ≡ tm4 ;; tm-to-literal (j + 3) (j + 4)
definition tm6 ≡ tm5 ;; tm-append (j + 2) (j + 3)
definition tm7 = tm6 ;; tm-setn (j + 3) 0
definition tm8 ≡ tm7 ;; tm-setn (j + 4) 0
definition tmL ≡ WHILE [] ; λrs. rs ! (j + 1) ≠ □ DO tm8 DONE
```

definition *tm9* ≡ *tmL* ;; *tm-cr* (*j* + *2*)
definition *tm10* ≡ *tm9* ;; *tm-erase-cr* (*j* + *1*)

lemma *tm10-eq-tm-relabel-clause*: *tm10* = *tm-relabel-clause j*
  unfolding *tm-relabel-clause-def tm3-def tmL-def tm5-def tm4-def tm1-def tm2-def tm6-def tm7-def tm8-def tm9-def tm10-def*
  by *simp*

context
  fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *σ clause* :: *nat list*
  assumes *clause*: ∀ *v*∈*set clause*. *v div 2* < *length σ*
  assumes *jk*: *0* < *j j* + *4* < *k length tps0* = *k*
  assumes *tps0*:
    tps0 ! *j* = ($\lfloor \sigma \rfloor_{NL}$, *1*)
    tps0 ! (*j* + *1*) = ($\lfloor clause \rfloor_{NL}$, *1*)
    tps0 ! (*j* + *2*) = ($\lfloor [] \rfloor_{NL}$, *1*)
    tps0 ! (*j* + *3*) = ($\lfloor 0 \rfloor_{N}$, *1*)
    tps0 ! (*j* + *4*) = ($\lfloor 0 \rfloor_{N}$, *1*)
begin

definition *tpsL t* ≡ *tps0*
  [*j* + *1* := *nltape′ clause t*,
   j + *2* := *nltape* (*take t* (*map literal-n* (*map* (*rename σ*) (*n-clause clause*))))]

lemma *tpsL-eq-tps0*: *tpsL 0* = *tps0*
  using *tpsL-def tps0 jk nllength-Nil* **by** (*metis One-nat-def list-update-id take0*)

definition *tps1 t* ≡ *tps0*
  [*j* + *1* := *nltape′ clause* (*Suc t*),
   j + *2* := *nltape* (*take t* (*map literal-n* (*map* (*rename σ*) (*n-clause clause*)))),
   j + *3* := ($\lfloor clause\ !\ t \rfloor_{N}$, *1*)]

lemma *tm1* [*transforms-intros*]:
  assumes *t* < *length clause*
    and *ttt* = *12* + *2* ∗ *nlength* (*clause* ! *t*)
  shows *transforms tm1* (*tpsL t*) *ttt* (*tps1 t*)
  unfolding *tm1-def*
proof (*tform tps*: *assms*(*1*) *tpsL-def tps1-def tps0 jk*)
  show *ttt* = *12* + *2* ∗ *nlength 0* + *2* ∗ *nlength* (*clause* ! *t*)
    using *assms*(*2*) **by** *simp*
qed

definition *tps2 t* ≡ *tps0*
  [*j* + *1* := *nltape′ clause* (*Suc t*),
   j + *2* := *nltape* (*take t* (*map literal-n* (*map* (*rename σ*) (*n-clause clause*)))),
   j + *3* := ($\lfloor clause\ !\ t \rfloor_{N}$, *1*),
   j + *4* := ($\lfloor (clause\ !\ t)\ mod\ 2 \rfloor_{N}$, *1*)]

lemma *tm2* [*transforms-intros*]:
  assumes *t* < *length clause*
    and *ttt* = *13* + *2* ∗ *nlength* (*clause* ! *t*)
  shows *transforms tm2* (*tpsL t*) *ttt* (*tps2 t*)
  unfolding *tm2-def* **by** (*tform tps*: *assms tps1-def tps2-def tps0 jk*)

definition *tps3 t* ≡ *tps0*
  [*j* + *1* := *nltape′ clause* (*Suc t*),
   j + *2* := *nltape* (*take t* (*map literal-n* (*map* (*rename σ*) (*n-clause clause*)))),
   j + *3* := ($\lfloor clause\ !\ t\ div\ 2 \rfloor_{N}$, *1*),
   j + *4* := ($\lfloor clause\ !\ t\ mod\ 2 \rfloor_{N}$, *1*)]

lemma *tm3* [*transforms-intros*]:
  assumes *t* < *length clause*
    and *ttt* = *16* + *4* ∗ *nlength* (*clause* ! *t*)

shows *transforms tm3* (*tpsL t*) *ttt* (*tps3 t*)
unfolding *tm3-def* **by** (*tform tps*: *assms*(*1*) *tps2-def tps3-def jk time*: *assms*(*2*))

definition *tps4 t* ≡ *tps0*
[*j* + *1* := *nltape′ clause* (*Suc t*),
j + *2* := *nltape* (*take t* (*map literal-n* (*map* (*rename* σ) (*n-clause clause*)))),
j + *3* := (⌊σ ! (*clause* ! *t div 2*)⌋$_N$, *1*),
j + *4* := (⌊*clause* ! *t mod 2*⌋$_N$, *1*)]

lemma *tm4* [*transforms-intros*]:
assumes *t* < *length clause*
and *ttt* = *28* + *4* ∗ *nlength* (*clause* ! *t*) + *18* ∗ (*nllength* σ)^2
shows *transforms tm4* (*tpsL t*) *ttt* (*tps4 t*)
unfolding *tm4-def*
proof (*tform tps*: *assms*(*1*) *tps0 tps3-def tps4-def jk clause time*: *assms*(*2*))
show *tps4 t* = (*tps3 t*)[*j* + *3* := (⌊σ ! (*clause* ! *t div 2*)⌋$_N$, *1*)]
unfolding *tps4-def tps3-def* **by** (*simp add*: *list-update-swap*[*of j* + *3*])
qed

definition *tps5 t* ≡ *tps0*
[*j* + *1* := *nltape′ clause* (*Suc t*),
j + *2* := *nltape* (*take t* (*map literal-n* (*map* (*rename* σ) (*n-clause clause*)))),
j + *3* := (⌊*2* ∗ (σ ! (*clause* ! *t div 2*)) + *clause* ! *t mod 2*⌋$_N$, *1*),
j + *4* := (⌊*clause* ! *t mod 2*⌋$_N$, *1*)]

lemma *tm5* [*transforms-intros*]:
assumes *t* < *length clause*
and *ttt* = *41* + *4* ∗ *nlength* (*clause* ! *t*) + *18* ∗ (*nllength* σ)^2 +
4 ∗ *nlength* (σ ! (*clause* ! *t div 2*))
shows *transforms tm5* (*tpsL t*) *ttt* (*tps5 t*)
unfolding *tm5-def* **by** (*tform tps*: *assms*(*1*) *tps0 tps4-def tps5-def jk time*: *assms*(*2*))

definition *tps6 t* ≡ *tps0*
[*j* + *1* := *nltape′ clause* (*Suc t*),
j + *2* := *nltape* (*take* (*Suc t*) (*map literal-n* (*map* (*rename* σ) (*n-clause clause*)))),
j + *3* := (⌊*2* ∗ (σ ! (*clause* ! *t div 2*)) + *clause* ! *t mod 2*⌋$_N$, *1*),
j + *4* := (⌊*clause* ! *t mod 2*⌋$_N$, *1*)]

lemma *tm6*:
assumes *t* < *length clause*
and *ttt* = *47* + *4* ∗ *nlength* (*clause* ! *t*) + *18* ∗ (*nllength* σ)^2 +
4 ∗ *nlength* (σ ! (*clause* ! *t div 2*)) +
2 ∗ *nlength* (*2* ∗ σ ! (*clause* ! *t div 2*) + *clause* ! *t mod 2*)
shows *transforms tm6* (*tpsL t*) *ttt* (*tps6 t*)
unfolding *tm6-def*
proof (*tform tps*: *assms*(*1*) *tps0 tps5-def tps6-def jk*)
have *2* ∗ σ ! (*clause* ! *t div 2*) + *clause* ! *t mod 2* =
(*literal-n* ∘ *rename* σ) (*n-literal* (*clause* ! *t*))
using *clause assms*(*1*) *literal-n-rename* **by** *simp*
then have *2* ∗ σ ! (*clause* ! *t div 2*) + *clause* ! *t mod 2* =
(*map* (*literal-n* ∘ *rename* σ) (*n-clause clause*)) ! *t*
using *assms*(*1*) **by** (*simp add*: *n-clause-def*)
then have *take t* (*map* (*literal-n* ∘ *rename* σ) (*n-clause clause*)) @
[*2* ∗ σ ! (*clause* ! *t div 2*) + *clause* ! *t mod 2*] =
take (*Suc t*) (*map* (*literal-n* ∘ *rename* σ) (*n-clause clause*))
by (*simp add*: *assms*(*1*) *n-clause-def take-Suc-conv-app-nth*)
then show *tps6 t* = (*tps5 t*)
[*j* + *2* := *nltape*
(*take t* (*map* (*literal-n* ∘ *rename* σ) (*n-clause clause*)) @
[*2* ∗ σ ! (*clause* ! *t div 2*) + *clause* ! *t mod 2*])]
unfolding *tps5-def tps6-def*
by (*simp only*: *list-update-overwrite list-update-swap-less*[*of j* + *2*]) *simp*
show *ttt* = *41* + *4* ∗ *nlength* (*clause* ! *t*) + *18* ∗ (*nllength* σ)^2 +

$4 * nlength\ (\sigma\ !\ (clause\ !\ t\ div\ 2)) +$
$(7 + nllength\ (take\ t\ (map\ (literal\text{-}n \circ rename\ \sigma)\ (n\text{-}clause\ clause))) -$
$Suc\ (nllength\ (take\ t\ (map\ (literal\text{-}n \circ rename\ \sigma)\ (n\text{-}clause\ clause)))) +$
$2 * nlength\ (2 * \sigma\ !\ (clause\ !\ t\ div\ 2) + clause\ !\ t\ mod\ 2))$
using *assms*(*2*) **by** *simp*
qed

lemma *nlength-σ1*:
assumes $t < length\ clause$
shows $nlength\ (clause\ !\ t) \leq nllength\ \sigma$
proof −
have $clause\ !\ t\ div\ 2 < length\ \sigma$
using *clause assms*(*1*) **by** *simp*
then have $nlength\ (clause\ !\ t\ div\ 2) < length\ \sigma$
using *nlength-le-n* **by** (*meson leD le-less-linear order.trans*)
then have $nlength\ (clause\ !\ t) \leq length\ \sigma$
using *canrepr-div-2* **by** *simp*
then show $nlength\ (clause\ !\ t) \leq nllength\ \sigma$
using *length-le-nllength* **by** (*meson dual-order.trans mult-le-mono2*)
qed

lemma *nlength-σ2*:
assumes $t < length\ clause$
shows $nlength\ (\sigma\ !\ (clause\ !\ t\ div\ 2)) \leq nllength\ \sigma$
using *assms clause member-le-nllength nth-mem* **by** *simp*

lemma *nlength-σ3*:
assumes $t < length\ clause$
shows $nlength\ (2 * \sigma\ !\ (clause\ !\ t\ div\ 2) + clause\ !\ t\ mod\ 2) \leq Suc\ (nllength\ \sigma)$
proof −
have $nlength\ (2 * \sigma\ !\ (clause\ !\ t\ div\ 2) + clause\ !\ t\ mod\ 2) \leq nlength\ (2 * \sigma\ !\ (clause\ !\ t\ div\ 2) + 1)$
using *nlength-mono* **by** *simp*
also have ... $\leq Suc\ (nlength\ (\sigma\ !\ (clause\ !\ t\ div\ 2)))$
using *nlength-times2plus1* **by** (*meson dual-order.trans*)
finally show *?thesis*
using *nlength-σ2 assms* **by** *fastforce*
qed

lemma *tm6′* [*transforms-intros*]:
assumes $t < length\ clause$
and $ttt = 49 + 28 * nllength\ \sigma\ \hat{}\ 2$
shows *transforms tm6* (*tpsL t*) *ttt* (*tps6 t*)
proof −
let $?ttt = 47 + 4 * nlength\ (clause\ !\ t) + 18 * (nllength\ \sigma)^2 +$
$4 * nlength\ (\sigma\ !\ (clause\ !\ t\ div\ 2)) +$
$2 * nlength\ (2 * \sigma\ !\ (clause\ !\ t\ div\ 2) + clause\ !\ t\ mod\ 2)$

have $?ttt \leq 47 + 4 * nllength\ \sigma + 18 * (nllength\ \sigma)^2 +$
$4 * nllength\ \sigma + 2 * Suc\ (nllength\ \sigma)$
using *nlength-σ1 nlength-σ3 nlength-σ2 assms*(*1*) **by** *fastforce*
also have ... $= 49 + 10 * nllength\ \sigma + 18 * (nllength\ \sigma)^2$
by *simp*
also have ... $\leq 49 + 10 * nllength\ \sigma\ \hat{}\ 2 + 18 * (nllength\ \sigma)^2$
using *linear-le-pow* **by** *simp*
also have ... $= 49 + 28 * nllength\ \sigma\ \hat{}\ 2$
by *simp*
finally have $?ttt \leq 49 + 28 * nllength\ \sigma\ \hat{}\ 2$ **.**
then show *?thesis*
using *tm6 assms transforms-monotone* **by** *blast*
qed

definition $tps7\ t \equiv tps0$
$[j + 1 := nltape'\ clause\ (Suc\ t),$

```
  j + 2 := nltape (take (Suc t) (map literal-n (map (rename σ) (n-clause clause)))),
  j + 4 := (⌊clause ! t mod 2⌋N, 1)]

lemma tm7:
  assumes t < length clause
   and ttt = 59 + 28 * nllength σ ^ 2 +
     2 * nlength (2 * σ ! (clause ! t div 2) + clause ! t mod 2)
  shows transforms tm7 (tpsL t) ttt (tps7 t)
  unfolding tm7-def
proof (tform tps: assms(1) tps0 tps6-def tps7-def jk time: assms(2))
  show tps7 t = (tps6 t)[j + 3 := (⌊0⌋N, 1)]
    using tps6-def tps7-def tps0 jk
    by (smt (verit) add-left-cancel list-update-id list-update-overwrite list-update-swap num.simps(8)
      numeral-eq-iff one-eq-numeral-iff semiring-norm(84))
qed

lemma tm7' [transforms-intros]:
  assumes t < length clause
   and ttt = 61 + 30 * nllength σ ^ 2
  shows transforms tm7 (tpsL t) ttt (tps7 t)
proof −
  let ?ttt = 59 + 28 * nllength σ ^ 2 +
     2 * nlength (2 * σ ! (clause ! t div 2) + clause ! t mod 2)
  have ?ttt ≤ 59 + 28 * nllength σ ^ 2 + 2 * Suc (nllength σ)
    using nlength-σ3 assms(1) by fastforce
  also have ... = 61 + 28 * nllength σ ^ 2 + 2 * nllength σ
    by simp
  also have ... ≤ 61 + 30 * nllength σ ^ 2
    using linear-le-pow by simp
  finally have ?ttt ≤ 61 + 30 * nllength σ ^ 2 .
  then show ?thesis
    using assms tm7 transforms-monotone by blast
qed

definition tps8 t ≡ tps0
  [j + 1 := nltape' clause (Suc t),
   j + 2 := nltape (take (Suc t) (map literal-n (map (rename σ) (n-clause clause))))]

lemma tm8:
  assumes t < length clause
    and ttt = 61 + 30 * (nllength σ)² + (10 + 2 * nlength (clause ! t mod 2))
  shows transforms tm8 (tpsL t) ttt (tps8 t)
  unfolding tm8-def
proof (tform tps: assms(1) tps0 tps7-def tps8-def jk time: assms(2))
  show tps8 t = (tps7 t)[j + 4 := (⌊0⌋N, 1)]
    using tps7-def tps8-def tps0 jk
    by (smt (verit) add-left-imp-eq list-update-id list-update-overwrite list-update-swap numeral-eq-iff
      numeral-eq-one-iff semiring-norm(85) semiring-norm(87))
qed

lemma tm8' [transforms-intros]:
  assumes t < length clause
    and ttt = 71 + 32 * (nllength σ)²
  shows transforms tm8 (tpsL t) ttt (tpsL (Suc t))
proof −
  have nlength (clause ! t mod 2) ≤ nllength σ
    using assms(1) nlength-σ1 by (meson mod-less-eq-dividend nlength-mono order.trans)
  then have nlength (clause ! t mod 2) ≤ nllength σ ^ 2
    using linear-le-pow by (metis nat-le-linear power2-nat-le-imp-le verit-la-disequality)
  then have 61 + 30 * (nllength σ)² + (10 + 2 * nlength (clause ! t mod 2)) ≤ ttt
    using assms(2) by simp
  then have transforms tm8 (tpsL t) ttt (tps8 t)
    using assms(1) tm8 transforms-monotone by blast
```

moreover have *tps8 t* = *tpsL* (*Suc t*)
  using *tps8-def tpsL-def* **by** *simp*
ultimately show *?thesis*
  by *simp*
qed

lemma *tmL* [*transforms-intros*]:
  assumes *ttt* = *length clause* ∗ (*73* + *32* ∗ (*nllength* σ)^2) + *1*
  shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsL* (*length clause*))
  unfolding *tmL-def*
proof (*tform*)
  let *?t* = *71* + *32* ∗ (*nllength* σ)^2
  show *read* (*tpsL t*) ! (*j* + *1*) ≠ □ **if** *t* < *length clause* **for** *t*
  proof −
    have *tpsL t* ! (*j* + *1*) = *nltape′ clause t*
      using *tpsL-def jk* **by** *simp*
    then show *?thesis*
      using *nltape′-tape-read that tapes-at-read′ tpsL-def jk*
      by (*smt* (*verit*) *Suc-eq-plus1 leD length-list-update less-add-same-cancel1 less-trans-Suc zero-less-numeral*)
  qed
  show ¬ *read* (*tpsL* (*length clause*)) ! (*j* + *1*) ≠ □
  proof −
    have *tpsL* (*length clause*) ! (*j* + *1*) = *nltape′ clause* (*length clause*)
      using *tpsL-def jk* **by** *simp*
    then show *?thesis*
      using *nltape′-tape-read tapes-at-read′ tpsL-def jk*
     by (*smt* (*verit*) *Suc-eq-plus1 length-list-update less-add-same-cancel1 less-or-eq-imp-le less-trans-Suc zero-less-numeral*)
  qed
  show *length clause* ∗ (*71* + *32* ∗ (*nllength* σ)^2 + *2*) + *1* ≤ *ttt*
    using *assms*(*1*) **by** *simp*
qed

lemma *tpsL-length*: *tpsL* (*length clause*) = *tps0*
  [*j* + *1* := *nltape′ clause* (*length clause*),
   j + *2* := *nltape* (*map literal-n* (*map* (*rename* σ) (*n-clause clause*)))]
  using *tpsL-def* **by** (*simp add*: *n-clause-def*)

definition *tps9* ≡ *tps0*
  [*j* + *1* := *nltape′ clause* (*length clause*),
   j + *2* := (⌊*map literal-n* (*map* (*rename* σ) (*n-clause clause*))⌋$_{NL}$, *1*)]

lemma *tm9*:
  assumes *ttt* = *4* + *length clause* ∗ (*73* + *32* ∗ (*nllength* σ)^2) +
      nllength (*map* (*literal-n* ∘ *rename* σ) (*n-clause clause*))
  shows *transforms tm9* (*tpsL 0*) *ttt tps9*
  unfolding *tm9-def*
proof (*tform tps*: *tps0 tps9-def tpsL-def jk tpsL-length*)
  show *clean-tape* (*tpsL* (*length clause*) ! (*j* + *2*))
    using *tpsL-def jk clean-tape-nlcontents tps0*(*3*) **by** *simp*
  show *ttt* = *length clause* ∗ (*73* + *32* ∗ (*nllength* σ)^2) + *1* +
    (*tpsL* (*length clause*) :#: (*j* + *2*) + *2*)
    using *n-clause-def assms jk tpsL-length* **by** *fastforce*
qed

lemma *tm9′* [*transforms-intros*]:
  assumes *ttt* = *4* + *2* ∗ *length clause* ∗ (*73* + *32* ∗ (*nllength* σ)^2)
  shows *transforms tm9 tps0 ttt tps9*
proof −
  let *?ttt* = *4* + *length clause* ∗ (*73* + *32* ∗ (*nllength* σ)^2) +
    nllength (*map* (*literal-n* ∘ *rename* σ) (*n-clause clause*))
  have *?ttt* ≤ *4* + *length clause* ∗ (*73* + *32* ∗ (*nllength* σ)^2) +
      length clause ∗ *Suc* (*nllength* σ)
    using *clause nllength-rename* **by** *simp*

also have ... $\leq$ *4* + *length clause* $*$ (*73* + *32* $*$ (*nllength* σ)^2) +
length clause $*$ (*Suc* (*nllength* σ ^ *2*))
by (*simp add*: *linear-le-pow*)
also have ... $\leq$ *4* + *length clause* $*$ (*73* + *32* $*$ (*nllength* σ)^2) +
length clause $*$ (*73* + *nllength* σ ^ *2*)
by (*metis One-nat-def Suc-eq-plus1 Suc-leI add.commute add-left-mono mult-le-mono2 zero-less-numeral*)
also have ... $\leq$ *4* + *length clause* $*$ (*73* + *32* $*$ (*nllength* σ)^2) +
length clause $*$ (*73* + *32* $*$ *nllength* σ ^ *2*)
by *simp*
also have ... = *4* + *2* $*$ *length clause* $*$ (*73* + *32* $*$ (*nllength* σ)^2)
by *simp*
finally have *?ttt* $\leq$ *4* + *2* $*$ *length clause* $*$ (*73* + *32* $*$ (*nllength* σ)^2) **.**
then show *?thesis*
using *tm9 assms transforms-monotone tpsL-eq-tps0* **by** *fastforce*
qed

definition *tps10* $\equiv$ *tps0*
[*j* + *1* := ($\lfloor[]\rfloor_{NL}$, *1*),
j + *2* := ($\lfloor$*map literal-n* (*map* (*rename* σ) (*n-clause clause*))$\rfloor_{NL}$, *1*)]

lemma *tm10*:
assumes *ttt* = *11* + *2* $*$ *length clause* $*$ (*73* + *32* $*$ (*nllength* σ)^2) + *3* $*$ *nllength clause*
shows *transforms tm10 tps0 ttt tps10*
unfolding *tm10-def*
proof (*tform tps*: *tps0 tps9-def jk*)
show *tps9* ::: (*j* + *1*) = $\lfloor$*numlist clause*$\rfloor$
using *tps9-def jk tps0*(*2*) *nlcontents-def* **by** *simp*
show *proper-symbols* (*numlist clause*)
using *proper-symbols-numlist* **by** *simp*
show *tps10* = *tps9*[*j* + *1* := ($\lfloor[]\rfloor$, *1*)]
by (*simp add*: *jk nlcontents-def tps0 tps10-def tps9-def list-update-swap numlist-Nil*)
show *ttt* = *4* + *2* $*$ *length clause* $*$ (*73* + *32* $*$ (*nllength* σ)^2) +
(*tps9* :#: (*j* + *1*) + *2* $*$ *length* (*numlist clause*) + *6*)
using *assms tps9-def jk nllength-def* **by** *simp*
qed

lemma *tm10'*:
assumes *ttt* = *11* + *64* $*$ *nllength clause* $*$ (*3* + (*nllength* σ)^2)
shows *transforms tm10 tps0 ttt tps10*
proof −
let *?ttt* = *11* + *2* $*$ *length clause* $*$ (*73* + *32* $*$ (*nllength* σ)^2) + *3* $*$ *nllength clause*
have *?ttt* $\leq$ *11* + *2* $*$ *nllength clause* $*$ (*73* + *32* $*$ (*nllength* σ)^2) + *3* $*$ *nllength clause*
by (*simp add*: *length-le-nllength*)
also have ... $\leq$ *11* + *2* $*$ *nllength clause* $*$ (*73* + *32* $*$ (*nllength* σ)^2) + *2* $*$ *2* $*$ *nllength clause*
by *simp*
also have ... = *11* + *2* $*$ *nllength clause* $*$ (*75* + *32* $*$ (*nllength* σ)^2)
by *algebra*
also have ... $\leq$ *11* + *2* $*$ *nllength clause* $*$ (*96* + *32* $*$ (*nllength* σ)^2)
by *simp*
also have ... = *11* + *2* $*$ *32* $*$ *nllength clause* $*$ (*3* + (*nllength* σ)^2)
by *simp*
also have ... = *11* + *64* $*$ *nllength clause* $*$ (*3* + (*nllength* σ)^2)
by *simp*
finally have *?ttt* $\leq$ *11* + *64* $*$ *nllength clause* $*$ (*3* + (*nllength* σ)^2) **.**
then show *?thesis*
using *tm10 assms transforms-monotone* **by** *blast*
qed

end

end

lemma *transforms-tm-relabel-clauseI* [*transforms-intros*]:

fixes *j* :: *tapeidx*
fixes *tps tps'* :: *tape list* **and** *ttt k* :: *nat* **and** *σ clause* :: *nat list*
assumes *0 < j j + 4 < k length tps = k*
and ∀ *v*∈*set clause. v div 2 < length σ*
assumes
tps ! j = (⌊*σ*⌋*NL*, *1*)
tps ! (*j* + *1*) = (⌊*clause*⌋*NL*, *1*)
tps ! (*j* + *2*) = (⌊[]⌋*NL*, *1*)
tps ! (*j* + *3*) = (⌊*0*⌋*N*, *1*)
tps ! (*j* + *4*) = (⌊*0*⌋*N*, *1*)
assumes *ttt* = *11* + *64* ∗ *nllength clause* ∗ (*3* + (*nllength σ*)^2)
assumes *tps'* = *tps*
[*j* + *1* := (⌊[]⌋*NL*, *1*),
j + *2* := (⌊*clause-n* (*map* (*rename σ*) (*n-clause clause*))⌋*NL*, *1*)]
shows *transforms* (*tm-relabel-clause j*) *tps ttt tps'*
proof −
interpret *loc*: *turing-machine-relabel-clause j* **.**
show *?thesis*
using *assms loc.tm10-eq-tm-relabel-clause loc.tps10-def loc.tm10' clause-n-def* **by** *simp*
qed

7.2.3 Relabeling CNF formulas

A Turing machine can relabel a CNF formula by relabeling each clause using the TM *tm-relabel-clause*. The next TM accepts a CNF formula as a list of lists of literals on tape j_1 and a substitution σ as a list of numbers on tape $j_2 + 1$. It outputs the relabeled formula on tape j_2, which initially must be empty.

definition *tm-relabel* :: *tapeidx* ⇒ *tapeidx* ⇒ *machine* **where**
tm-relabel j1 j2 ≡
WHILE [] ; *λrs. rs ! j1* ≠ □ *DO*
tm-nextract ♯ *j1* (*j2* + *2*) ;;
tm-relabel-clause (*j2* + *1*) ;;
tm-appendl j2 (*j2* + *3*) ;;
tm-erase-cr (*j2* + *3*)
DONE ;;
tm-cr j1 ;;
tm-cr j2

lemma *tm-relabel-tm*:
assumes *G* ≥ *6* **and** *j2* + *5* < *k* **and** *0* < *j1* **and** *j1* < *j2*
shows *turing-machine k G* (*tm-relabel j1 j2*)
unfolding *tm-relabel-def*
using *assms tm-cr-tm tm-nextract-tm tm-appendl-tm tm-relabel-clause-tm Nil-tm tm-erase-cr-tm turing-machine-loop-turing-machi*
by *simp*

locale *turing-machine-relabel* =
fixes *j1 j2* :: *tapeidx*
begin

definition *tmL1* ≡ *tm-nextract* ♯ *j1* (*j2* + *2*)
definition *tmL2* ≡ *tmL1* ;; *tm-relabel-clause* (*j2* + *1*)
definition *tmL3* ≡ *tmL2* ;; *tm-appendl j2* (*j2* + *3*)
definition *tmL4* ≡ *tmL3* ;; *tm-erase-cr* (*j2* + *3*)
definition *tmL* ≡ *WHILE* [] ; *λrs. rs ! j1* ≠ □ *DO tmL4 DONE*
definition *tm2* ≡ *tmL* ;; *tm-cr j1*
definition *tm3* ≡ *tm2* ;; *tm-cr j2*

lemma *tm3-eq-tm-relabel*: *tm3* = *tm-relabel j1 j2*
unfolding *tm-relabel-def tm2-def tmL-def tmL4-def tmL3-def tmL2-def tmL1-def tm3-def* **by** *simp*

context
fixes *tps0* :: *tape list* **and** *k* :: *nat* **and** *σ* :: *nat list* **and** *φ* :: *formula*
assumes *variables*: *variables φ* ⊆ {..<*length σ*}
assumes *jk*: *0* < *j1 j1* < *j2 j2* + *5* < *k length tps0* = *k*

```
  assumes tps0:
    tps0 ! j1 = (⌊formula-n φ⌋_NLL, 1)
    tps0 ! j2 = (⌊[]⌋_NLL, 1)
    tps0 ! (j2 + 1) = (⌊σ⌋_NL, 1)
    tps0 ! (j2 + 2) = (⌊[]⌋_NL, 1)
    tps0 ! (j2 + 3) = (⌊[]⌋_NL, 1)
    tps0 ! (j2 + 4) = (⌊0⌋_N, 1)
    tps0 ! (j2 + 5) = (⌊0⌋_N, 1)
begin

abbreviation nφ :: nat list list where
  nφ ≡ formula-n φ

definition tpsL :: nat ⇒ tape list where
  tpsL t ≡ tps0
    [j1 := nlltape' nφ t,
     j2 := nlltape (formula-n (take t (relabel σ φ)))]

lemma tpsL-eq-tps0: tpsL 0 = tps0
  using tps0 tpsL-def formula-n-def nlllength-def numlist-Nil numlist-def numlistlist-def
  by (metis One-nat-def list.map(1) list.size(3) list-update-id take0)

definition tpsL1 :: nat ⇒ tape list where
  tpsL1 t ≡ tps0
    [j1 := nlltape' nφ (Suc t),
     j2 := nlltape (formula-n (take t (relabel σ φ))),
     j2 + 2 := (⌊nφ ! t⌋_NL, 1)]

lemma tmL1 [transforms-intros]:
  assumes ttt = 12 + 2 ∗ nllength (nφ ! t)
    and t < length φ
  shows transforms tmL1 (tpsL t) ttt (tpsL1 t)
  unfolding tmL1-def
proof (tform tps: tps0 tpsL-def tpsL1-def jk)
  show t < length nφ
    using assms(2) formula-n-def by simp
  show tpsL1 t = (tpsL t)
      [j1 := nlltape' nφ (Suc t),
       j2 + 2 := (⌊nφ ! t⌋_NL, 1)]
    using tpsL1-def tpsL-def by (simp add: jk list-update-swap-less)
  show ttt = 12 + 2 ∗ nllength [] + 2 ∗ nllength (nφ ! t)
    using assms(1) by simp
qed

definition tpsL2 :: nat ⇒ tape list where
  tpsL2 t ≡ tps0
    [j1 := nlltape' nφ (Suc t),
     j2 := nlltape (formula-n (take t (relabel σ φ))),
     j2 + 2 := (⌊[]⌋_NL, 1),
     j2 + 3 := (⌊clause-n (map (rename σ) (n-clause (nφ ! t)))⌋_NL, 1)]

lemma tmL2 [transforms-intros]:
  assumes ttt = 23 + 2 ∗ nllength (nφ ! t) + 64 ∗ nllength (nφ ! t) ∗ (3 + (nllength σ)^2)
    and t < length φ
  shows transforms tmL2 (tpsL t) ttt (tpsL2 t)
  unfolding tmL2-def
proof (tform tps: assms(2) tps0 tpsL1-def jk)
  show ∀ v∈set (nφ ! t). v div 2 < length σ
    using variables-σ variables assms(2) by (metis formula-n-def nth-map nth-mem)
  have tpsL1 t ! (j2 + (1 + 2)) = (⌊[]⌋_NL, 1)
    using tpsL1-def jk tps0 by (simp add: numeral-3-eq-3)
  then show tpsL1 t ! (j2 + 1 + 2) = (⌊[]⌋_NL, 1)
    by (metis add.assoc)
```

have *tpsL1 t* ! (*j2* + (*1* + *3*)) = ($\lfloor 0 \rfloor_N$, *1*)
using *tpsL1-def jk tps0* **by** *simp*
then show *tpsL1 t* ! (*j2* + *1* + *3*) = ($\lfloor 0 \rfloor_N$, *1*)
by (*metis add.assoc*)
have *tpsL1 t* ! (*j2* + (*1* + *4*)) = ($\lfloor 0 \rfloor_N$, *1*)
using *tpsL1-def jk tps0* **by** *simp*
then show *tpsL1 t* ! (*j2* + *1* + *4*) = ($\lfloor 0 \rfloor_N$, *1*)
by (*metis add.assoc*)
have *tpsL2 t* = (*tpsL1 t*)
[*j2* + (*1* + *1*) := ($\lfloor [] \rfloor_{NL}$, *1*),
j2 + (*1* + *2*) := ($\lfloor$*clause-n* (*map* (*rename* σ) (*n-clause* (*n*φ ! *t*)))$\rfloor_{NL}$, *1*)]
using *jk tps0 tpsL1-def tpsL2-def* **by** (*simp add*: *numeral-3-eq-3*)
then show *tpsL2 t* = (*tpsL1 t*)
[*j2* + *1* + *1* := ($\lfloor [] \rfloor_{NL}$, *1*),
j2 + *1* + *2* := ($\lfloor$*clause-n* (*map* (*rename* σ) (*n-clause* (*n*φ ! *t*)))$\rfloor_{NL}$, *1*)]
by (*metis add.assoc*)
show *ttt* = *12* + *2* $*$ *nllength* (*n*φ ! *t*) +
(*11* + *64* $*$ *nllength* (*n*φ ! *t*) $*$ (*3* + (*nllength* σ)^2))
using *assms*(*1*) **by** *simp*
qed

definition *tpsL3* :: *nat* $\Rightarrow$ *tape list* **where**
tpsL3 t $\equiv$ *tps0*
[*j1* := *nlltape*$'$ *n*φ (*Suc t*),
j2 := *nlltape*
(*formula-n* (*take t* (*relabel* σ φ)) @ [*clause-n* (*map* (*rename* σ) (*n-clause* (*n*φ ! *t*)))]),
j2 + *2* := ($\lfloor [] \rfloor_{NL}$, *1*),
j2 + *3* := ($\lfloor$*clause-n* (*map* (*rename* σ) (*n-clause* (*n*φ ! *t*)))$\rfloor_{NL}$, *1*)]

lemma *tmL3* [*transforms-intros*]:
assumes *ttt* = *29* + *2* $*$ *nllength* (*n*φ ! *t*) +
64 $*$ *nllength* (*n*φ ! *t*) $*$ (*3* + (*nllength* σ)^2) +
2 $*$ *nllength* (*clause-n* (*map* (*rename* σ) (*n-clause* (*n*φ ! *t*))))
and *t* < *length* φ
shows *transforms tmL3* (*tpsL t*) *ttt* (*tpsL3 t*)
unfolding *tmL3-def*
proof (*tform tps*: *assms*(*2*) *tps0 tpsL2-def jk*)
show *tpsL3 t* = (*tpsL2 t*)
[*j2* := *nlltape* (*formula-n* (*take t* (*relabel* σ φ)) @ [*clause-n* (*map* (*rename* σ) (*n-clause* (*n*φ ! *t*)))])]
unfolding *tpsL3-def tpsL2-def* **by** (*simp add*: *list-update-swap-less*[*of j2*])
show *ttt* = *23* + *2* $*$ *nllength* (*n*φ ! *t*) + *64* $*$ *nllength* (*n*φ ! *t*) $*$ (*3* + (*nllength* σ)^2) +
(*7* + *nlllength* (*formula-n* (*take t* (*relabel* σ φ))) $-$ *Suc* (*nlllength* (*formula-n* (*take t* (*relabel* σ φ)))) +
2 $*$ *nllength* (*clause-n* (*map* (*rename* σ) (*n-clause* (*n*φ ! *t*)))))
using *assms*(*1*) **by** *simp*
qed

definition *tpsL4* :: *nat* $\Rightarrow$ *tape list* **where**
tpsL4 t $\equiv$ *tps0*
[*j1* := *nlltape*$'$ *n*φ (*Suc t*),
j2 := *nlltape*
(*formula-n* (*take t* (*relabel* σ φ)) @ [*clause-n* (*map* (*rename* σ) (*n-clause* (*n*φ ! *t*)))]),
j2 + *2* := ($\lfloor [] \rfloor_{NL}$, *1*)]

lemma *tmL4*:
assumes *ttt* = *36* + *2* $*$ *nllength* (*n*φ ! *t*) +
64 $*$ *nllength* (*n*φ ! *t*) $*$ (*3* + (*nllength* σ)^2) +
4 $*$ *nllength* (*clause-n* (*map* (*rename* σ) (*n-clause* (*n*φ ! *t*))))
and *t* < *length* φ
shows *transforms tmL4* (*tpsL t*) *ttt* (*tpsL4 t*)
unfolding *tmL4-def*
proof (*tform tps*: *assms*(*2*) *tps0 tpsL3-def jk*)
let *?zs* = *numlist* (*clause-n* (*map* (*rename* σ) (*n-clause* (*n*φ ! *t*))))
show *tpsL3 t* ::: (*j2* + *3*) = $\lfloor$*?zs*$\rfloor$

using *tpsL3-def nlcontents-def jk* **by** *simp*
show *proper-symbols ?zs*
using *proper-symbols-numlist* **by** *simp*
have ∗: *j1* ≠ *j2* + *3*
using *jk* **by** *simp*
have ⌊[]⌋ = ⌊[]⌋$_{NL}$
using *nlcontents-def numlist-Nil* **by** *simp*
then show *tpsL4 t* = (*tpsL3 t*)[*j2* + *3* := (⌊[]⌋, *1*)]
using *tpsL3-def tpsL4-def tps0 jk list-update-id*[*of tps0 j2*+*3*]
by (*simp add*: *list-update-swap*[*OF* ∗] *list-update-swap*[*of - j2* + *3*])
show *ttt* = *29* + *2* ∗ *nllength* (*nφ* ! *t*) + *64* ∗ *nllength* (*nφ* ! *t*) ∗ (*3* + (*nllength σ*)^2) +
2 ∗ *nllength* (*clause-n* (*map* (*rename σ*) (*n-clause* (*nφ* ! *t*)))) +
(*tpsL3 t* :#: (*j2* + *3*) + *2* ∗ *length* (*numlist* (*clause-n* (*map* (*rename σ*) (*n-clause* (*nφ* ! *t*))))) + *6*)
using *assms*(*1*) *tpsL3-def jk nllength-def* **by** *simp*
qed

lemma *nllength-1*:
assumes *t* < *length φ*
shows *nllength* (*nφ* ! *t*) ≤ *nlllength nφ*
using *assms formula-n-def nlllength-take* **by** (*metis le-less-linear length-map less-trans not-add-less2*)

lemma *nllength-2*:
assumes *t* < *length φ*
shows *nllength* (*clause-n* (*map* (*rename σ*) (*n-clause* (*nφ* ! *t*)))) ≤ *nlllength* (*formula-n* (*relabel σ φ*))
(**is** *?l* ≤ *?r*)
proof −
have *?l* = *nllength* (*clause-n* (*map* (*rename σ*) (*φ* ! *t*)))
using *assms* **by** (*simp add*: *formula-n-def n-clause-n*)
moreover have *clause-n* (*map* (*rename σ*) (*φ* ! *t*)) ∈ *set* (*formula-n* (*relabel σ φ*))
using *assms formula-n-def relabel-def* **by** *simp*
ultimately show *?thesis*
using *member-le-nlllength-1* **by** *fastforce*
qed

definition *tpsL4′ t* ≡ *tps0*
[*j1* := *nlltape′ nφ* (*Suc t*),
j2 := *nlltape* (*formula-n* (*take* (*Suc t*) (*relabel σ φ*)))]

lemma *tpsL4*:
assumes *t* < *length φ*
shows *tpsL4 t* = *tpsL4′ t*
proof −
have *formula-n* (*take t* (*relabel σ φ*)) @ [*clause-n* (*map* (*rename σ*) (*n-clause* (*nφ* ! *t*)))] =
formula-n (*take* (*Suc t*) (*relabel σ φ*))
using *assms formula-n-def relabel-def* **by** (*simp add*: *n-clause-n take-Suc-conv-app-nth*)
then show *?thesis*
using *tpsL4-def tpsL4′-def jk tps0*
by (*smt* (*verit*, *del-insts*) *Suc-1 add-Suc-right add-cancel-left-right less-SucI*
list-update-id list-update-swap not-less-eq numeral-1-eq-Suc-0 numeral-One)
qed

lemma *tpsL4′-eq-tpsL*: *tpsL4′ t* = *tpsL* (*Suc t*)
using *tpsL-def tpsL4′-def* **by** *simp*

lemma *tmL4′* [*transforms-intros*]:
assumes *ttt* = *36* + *65* ∗ *nlllength nφ* ∗ (*3* + (*nllength σ*)^2) + *4* ∗ *nlllength* (*formula-n* (*relabel σ φ*))
and *t* < *length φ*
shows *transforms tmL4* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
proof −
let *?ttt* = *36* + *2* ∗ *nllength* (*nφ* ! *t*) +
64 ∗ *nllength* (*nφ* ! *t*) ∗ (*3* + (*nllength σ*)^2) +
4 ∗ *nllength* (*clause-n* (*map* (*rename σ*) (*n-clause* (*nφ* ! *t*))))
have *?ttt* ≤ *36* + *2* ∗ *nlllength nφ* +

64 ∗ nlllength nφ ∗ (3 + (nllength σ)^2) +
4 ∗ nllength (clause-n (map (rename σ) (n-clause (nφ ! t))))
using nllength-1 assms(2) add-le-mono add-le-mono1 mult-le-mono1 mult-le-mono2 nat-add-left-cancel-le
by metis
also have ... ≤ 36 + 2 ∗ nlllength nφ +
64 ∗ nlllength nφ ∗ (3 + (nllength σ)^2) +
4 ∗ nlllength (formula-n (relabel σ φ))
using nllength-2 assms(2) **by** simp
also have ... ≤ 36 + 65 ∗ nlllength nφ ∗ (3 + (nllength σ)^2) + 4 ∗ nlllength (formula-n (relabel σ φ))
by simp
finally have ?ttt ≤ 36 + 65 ∗ nlllength nφ ∗ (3 + (nllength σ)^2) + 4 ∗ nlllength (formula-n (relabel σ φ)) **.**
then have transforms tmL4 (tpsL t) ttt (tpsL4 t)
using assms tmL4 transforms-monotone **by** blast
then show ?thesis
using assms(2) tpsL4'-eq-tpsL tpsL4 **by** simp
qed

lemma tmL:
assumes ttt = length φ ∗ (38 + 65 ∗ nlllength nφ ∗ (3 + (nllength σ)^2) + 4 ∗ nlllength (formula-n (relabel σ φ))) + 1
shows transforms tmL (tpsL 0) ttt (tpsL (length φ))
unfolding tmL-def
proof (tform)
let ?t = 36 + 65 ∗ nlllength nφ ∗ (3 + (nllength σ)^2) + 4 ∗ nlllength (formula-n (relabel σ φ))
show ¬ read (tpsL (length φ)) ! j1 ≠ □
proof −
have tpsL (length φ) ! j1 = nlltape' nφ (length nφ)
using tpsL-def jk formula-n-def **by** simp
then show ?thesis
using nlltape'-tape-read[of nφ length nφ] tapes-at-read'[of j1 tpsL (length φ)] tpsL-def jk
by simp
qed
show read (tpsL t) ! j1 ≠ □ **if** t < length φ **for** t
proof −
have tpsL t ! j1 = nlltape' nφ t
using tpsL-def jk **by** simp
then show ?thesis
using that formula-n-def nlltape'-tape-read[of nφ t] tapes-at-read'[of j1 tpsL t] tpsL-def jk
by simp
qed
show length φ ∗ (?t + 2) + 1 ≤ ttt
using assms(1) **by** simp
qed

lemma tmL' [transforms-intros]:
assumes ttt = 107 ∗ nlllength nφ ^ 2 ∗ (3 + nllength σ ^ 2) + 1
shows transforms tmL (tpsL 0) ttt (tpsL (length φ))
proof −
let ?ttt = length φ ∗ (38 + 65 ∗ nlllength nφ ∗ (3 + (nllength σ)^2) + 4 ∗ nlllength (formula-n (relabel σ φ))) + 1
have ?ttt ≤ length φ ∗ (38 + 65 ∗ nlllength nφ ∗ (3 + (nllength σ)^2) + 4 ∗ (Suc (nllength σ) ∗ nlllength nφ)) + 1
using nlllength-relabel-variables variables **by** fastforce
also have ... ≤ length φ ∗ (38 + 65 ∗ nlllength nφ ∗ (3 + (nllength σ)^2) + 4 ∗ ((3 + nllength σ) ∗ nlllength nφ)) + 1
proof −
have Suc (nllength σ) ≤ 3 + nllength σ
by simp
then show ?thesis
using add-le-mono le-refl mult-le-mono **by** presburger
qed
also have ... ≤ length φ ∗ (38 + 65 ∗ nlllength nφ ∗ (3 + (nllength σ)^2) + 4 ∗ ((3 + nllength σ ^ 2) ∗ nlllength nφ)) + 1

using *linear-le-pow* **by** *simp*
also have ... = *length* φ ∗ (*38* + *69* ∗ *nlllength* $n\varphi$ ∗ (*3* + (*nllength* σ)^2)) + *1*
by *simp*
also have ... ≤ *nlllength* $n\varphi$ ∗ (*38* + *69* ∗ *nlllength* $n\varphi$ ∗ (*3* + (*nllength* σ)^2)) + *1*
using *length-le-nlllength formula-n-def* **by** (*metis add-mono-thms-linordered-semiring*(*3*) *length-map mult-le-mono1*)
also have ... = *38* ∗ *nlllength* $n\varphi$ + (*69* ∗ *nlllength* $n\varphi$ ^ *2* ∗ (*3* + (*nllength* σ)^2)) + *1*
by *algebra*
also have ... ≤ *38* ∗ *nlllength* $n\varphi$ ^ *2* ∗ (*3* + *nllength* σ ^ *2*) + (*69* ∗ *nlllength* $n\varphi$ ^ *2* ∗ (*3* + (*nllength* σ)^2)) + *1*
proof −
have *nlllength* $n\varphi$ ≤ *nlllength* $n\varphi$ ^ *2* ∗ (*3* + *nllength* σ ^ *2*)
using *linear-le-pow* **by** (*metis One-nat-def Suc-leI add-gr-0 mult-le-mono nat-mult-1-right zero-less-numeral*)
then show *?thesis*
by *simp*
qed
also have ... = *107* ∗ *nlllength* $n\varphi$ ^ *2* ∗ (*3* + *nllength* σ ^ *2*) + *1*
by *simp*
finally have *?ttt* ≤ *107* ∗ *nlllength* $n\varphi$ ^ *2* ∗ (*3* + *nllength* σ ^ *2*) + *1* **.**
then show *?thesis*
using *tmL assms transforms-monotone* **by** *blast*
qed

definition *tps1* :: *tape list* **where**
tps1 ≡ *tps0*
[*j1* := *nlltape′* $n\varphi$ (*length* φ),
j2 := *nlltape* (*formula-n* (*relabel* σ φ))]

lemma *tps1-eq-tpsL*: *tps1* = *tpsL* (*length* φ)
using *tps1-def tpsL-def jk tps0 relabel-def* **by** *simp*

definition *tps2* ≡ *tps0*
[*j2* := *nlltape* (*formula-n* (*relabel* σ φ))]

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *Suc* (*107* ∗ (*nlllength* $n\varphi$)^2 ∗ (*3* + (*nllength* σ)^2)) +
Suc (*Suc* (*Suc* (*nlllength* $n\varphi$)))
shows *transforms tm2* (*tpsL 0*) *ttt tps2*
unfolding *tm2-def*
proof (*tform tps*: *tps0 tpsL-def tps1-def jk*)
have ∗: *tpsL* (*length* φ) ! *j1* = *nlltape′* $n\varphi$ (*length* φ)
using *tpsL-def jk* **by** *simp*
then show *clean-tape* (*tpsL* (*length* φ) ! *j1*)
using *clean-tape-nllcontents* **by** *simp*
have *tpsL* (*length* φ) ! *j1* |#=| *1* = *nlltape′* $n\varphi$ *0*
using ∗ **by** *simp*
then show *tps2* = (*tpsL* (*length* φ))[*j1* := *tpsL* (*length* φ) ! *j1* |#=| *1*]
using *tps0 jk tps2-def tps1-eq-tpsL tps1-def*
by (*metis* (*no-types, lifting*) *One-nat-def list-update-id list-update-overwrite list-update-swap-less nlllength-Nil take0*)
show *ttt* = *107* ∗ (*nlllength* $n\varphi$)^2 ∗ (*3* + (*nllength* σ)^2) + *1* +
(*tpsL* (*length* φ) :#: *j1* + *2*)
using *assms tpsL-def jk formula-n-def* **by** *simp*
qed

definition *tps3* ≡ *tps0*
[*j2* := *nlltape′* (*formula-n* (*relabel* σ φ)) *0*]

lemma *tm3*:
assumes *ttt* = *7* + (*107* ∗ (*nlllength* $n\varphi$)^2 ∗ (*3* + (*nllength* σ)^2)) +
nlllength $n\varphi$ + *nlllength* (*formula-n* (*relabel* σ φ))
shows *transforms tm3* (*tpsL 0*) *ttt tps3*
unfolding *tm3-def*
proof (*tform tps*: *assms tps0 tps2-def tps3-def jk*)

show *clean-tape* (*tps2* ! *j2*)
using *tps2-def jk clean-tape-nllcontents* **by** *simp*
qed

lemma *tm3'* [*transforms-intros*]:
assumes $ttt = 7 + (108 * (nlllength\ n\varphi)^2 * (3 + (nllength\ \sigma)^2))$
shows *transforms tm3 tps0 ttt tps3*
proof −
let $?ttt = 7 + (107 * (nlllength\ n\varphi)^2 * (3 + (nllength\ \sigma)^2)) +$
$nlllength\ n\varphi + nlllength\ (formula\text{-}n\ (relabel\ \sigma\ \varphi))$
have $?ttt \leq 7 + (107 * (nlllength\ n\varphi)^2 * (3 + (nllength\ \sigma)^2)) +$
$nlllength\ n\varphi + Suc\ (nllength\ \sigma) * nlllength\ n\varphi$
using *variables nlllength-relabel-variables* **by** *simp*
also have $... = 7 + (107 * (nlllength\ n\varphi)^2 * (3 + (nllength\ \sigma)^2)) +$
$(2 + nllength\ \sigma) * nlllength\ n\varphi$
by *simp*
also have $... \leq 7 + (107 * (nlllength\ n\varphi)^2 * (3 + (nllength\ \sigma)^2)) +$
$(2 + nllength\ \sigma\ \hat{}\ 2) * nlllength\ n\varphi$
using *linear-le-pow* **by** *simp*
also have $... \leq 7 + (107 * (nlllength\ n\varphi)^2 * (3 + (nllength\ \sigma)^2)) + (3 + nllength\ \sigma\ \hat{}\ 2) * nlllength\ n\varphi$
by (*metis add-2-eq-Suc add-Suc-shift add-mono-thms-linordered-semiring*(*2*) *le-add2 mult.commute mult-le-mono2 numeral-3-eq-3*)
also have $... \leq 7 + (107 * (nlllength\ n\varphi)^2 * (3 + (nllength\ \sigma)^2)) +$
$(3 + nllength\ \sigma\ \hat{}\ 2) * nlllength\ n\varphi\ \hat{}\ 2$
using *linear-le-pow* **by** *simp*
also have $... = 7 + (108 * (nlllength\ n\varphi)^2 * (3 + (nllength\ \sigma)^2))$
by *simp*
finally have $?ttt \leq 7 + (108 * (nlllength\ n\varphi)^2 * (3 + (nllength\ \sigma)^2))$ **.**
then show *?thesis*
using *tm3 assms tpsL-eq-tps0 transforms-monotone* **by** *simp*
qed

end

end

lemma *transforms-tm-relabelI* [*transforms-intros*]:
fixes *j1 j2* :: *tapeidx*
fixes *tps tps'* :: *tape list* **and** *ttt k* :: *nat* **and** σ :: *nat list* **and** φ :: *formula*
assumes $0 < j1$ **and** $j1 < j2$ **and** $j2 + 5 < k$ **and** *length tps* = *k*
and *variables* $\varphi \subseteq \{..<length\ \sigma\}$
assumes
$tps\ !\ j1 = (\lfloor formula\text{-}n\ \varphi \rfloor_{NLL},\ 1)$
$tps\ !\ j2 = (\lfloor [] \rfloor_{NLL},\ 1)$
$tps\ !\ (j2 + 1) = (\lfloor \sigma \rfloor_{NL},\ 1)$
$tps\ !\ (j2 + 2) = (\lfloor [] \rfloor_{NL},\ 1)$
$tps\ !\ (j2 + 3) = (\lfloor [] \rfloor_{NL},\ 1)$
$tps\ !\ (j2 + 4) = (\lfloor 0 \rfloor_{N},\ 1)$
$tps\ !\ (j2 + 5) = (\lfloor 0 \rfloor_{N},\ 1)$
assumes *tps'* = *tps*
[$j2 := nlltape'\ (formula\text{-}n\ (relabel\ \sigma\ \varphi))\ 0$]
assumes $ttt = 7 + (108 * (nlllength\ (formula\text{-}n\ \varphi))^2 * (3 + (nllength\ \sigma)^2))$
shows *transforms* (*tm-relabel j1 j2*) *tps ttt tps'*
proof −
interpret *loc*: *turing-machine-relabel j1 j2* **.**
show *?thesis*
using *assms loc.tm3-eq-tm-relabel loc.tm3' loc.tps3-def* **by** *simp*
qed

7.3 Listing the head positions of a Turing machine

The formulas χ_t used for Φ_9 require the functions *inputpos* and *prev*, or more precisely the substitutions ϱ_t and ϱ'_t do.

In this section we build a TM that simulates a two-tape TM M on some input until it halts. During the simulation it creates two lists: one with the sequence of head positions on M's input tape and one with the sequence of head positions on M's output tape. The first list directly provides *inputpos*; the second list allows the computation of *prev* using the TM *tm-prev*.

7.3.1 Simulating and logging head movements

At the core of the simulation is the following Turing command. It interprets the tapes $j+7$ and $j+8$ as input tape and output tape of a two-tape Turing machine M and the symbol in the first cell of tape $j+6$ as the state of M. It then applies the actions of M in this configuration to the tapes $j+7$ and $j+8$ and adapts the state on tape $j+6$ accordingly. On top of that it writes ("logs") to tape j the direction in which M's input tape head has moved and to tape $j+3$ the direction in which M's work tape head has moved.

The head movement directions are encoded by the symbols $\Box$, $\triangleright$, and **0** for Left, Stay, and Right, respectively.

The command is parameterized by the TM M and its alphabet size G (and as usual the tape index j). The command does nothing if the state on tape $j+6$ is the halting state or if the symbol read from the simulated tape $j+7$ or $j+8$ is outside the alphabet G.

```
definition direction-to-symbol :: direction ⇒ symbol where
  direction-to-symbol d ≡ (case d of Left ⇒ □ | Stay ⇒ ▷ | Right ⇒ 0)

lemma direction-to-symbol-less: direction-to-symbol d < 3
  using direction-to-symbol-def by (cases d) simp-all

definition cmd-simulog :: nat ⇒ machine ⇒ tapeidx ⇒ command where
  cmd-simulog G M j rs ≡
   (1,
    if rs ! (j + 6) ≥ length M ∨ rs ! (j + 7) ≥ G ∨ rs ! (j + 8) ≥ G
    then map (λz. (z, Stay)) rs
    else
      map (λi. let sas = (M ! (rs ! (j + 6))) [rs ! (j + 7), rs ! (j + 8)] in
          if i = j then (direction-to-symbol (sas [~] 0), Stay)
          else if i = j + 3 then (direction-to-symbol (sas [~] 1), Stay)
          else if i = j + 6 then (fst sas, Stay)
          else if i = j + 7 then sas [!] 0
          else if i = j + 8 then sas [!] 1
          else (rs ! i, Stay))
        [0..<length rs])

lemma turing-command-cmd-simulog:
  fixes G H :: nat
  assumes turing-machine 2 G M and k ≥ j + 9 and j > 0 and H ≥ Suc (length M) and H ≥ G
  shows turing-command k 1 H (cmd-simulog G M j)
proof
  show ⋀gs. length gs = k ⟹ length ([!!] cmd-simulog G M j gs) = length gs
    using cmd-simulog-def by simp
  have G: H ≥ 4
    using assms(1,5) turing-machine-def by simp
  show cmd-simulog G M j rs [.] i < H
    if length rs = k and (⋀i. i < length rs ⟹ rs ! i < H) and i < length rs
    for rs i
  proof (cases rs ! (j + 6) ≥ length M ∨ rs ! (j + 7) ≥ G ∨ rs ! (j + 8) ≥ G)
    case True
    then show ?thesis
      using assms that cmd-simulog-def by simp
```

next
  case *False*
  then have *inbounds*: $rs ! (j + 6) < length\ M$
    by *simp*
  let $?cmd = M ! (rs ! (j + 6))$
  let $?gs = [rs ! (j + 7), rs ! (j + 8)]$
  let $?sas = ?cmd\ ?gs$
  have *lensas*: $length\ (snd\ ?sas) = 2$
    using *assms*(*1*) *inbounds turing-commandD*(*1*)
    by (*metis length-Cons list.size*(*3*) *numeral-2-eq-2 turing-machineD*(*3*))
  consider
      $i = j$
    | $i = j + 3$
    | $i = j + 6$
    | $i = j + 7$
    | $i = j + 8$
    | $i \neq j \wedge i \neq j + 3 \wedge i \neq j + 6 \wedge i \neq j + 7 \wedge i \neq j + 8$
    by *linarith*
  then show *?thesis*
  proof (*cases*)
    case *1*
    then have *cmd-simulog G M j rs* [!] $i =$ (*direction-to-symbol* (*?sas* [~] *0*), *Stay*)
      using *cmd-simulog-def False that* **by** *simp*
    then have *cmd-simulog G M j rs* [.] $i < 3$
      using *direction-to-symbol-less* **by** *simp*
    then show *?thesis*
      using *G* **by** *simp*
  next
    case *2*
    then have *cmd-simulog G M j rs* [!] $i =$ (*direction-to-symbol* (*?sas* [~] *1*), *Stay*)
      using *cmd-simulog-def False that* **by** *simp*
    then have *cmd-simulog G M j rs* [.] $i < 3$
      using *direction-to-symbol-less* **by** *simp*
    then show *?thesis*
      using *G* **by** *simp*
  next
    case *3*
    then have *cmd-simulog G M j rs* [!] $i =$ (*fst ?sas*, *Stay*)
      using *cmd-simulog-def False that* **by** *simp*
    then have *cmd-simulog G M j rs* [.] $i \leq length\ M$
      using *assms inbounds turing-commandD*(*4*) *turing-machineD*(*3*)
      by (*metis One-nat-def Suc-1 fst-conv length-Cons list.size*(*3*))
    then show *?thesis*
      using *assms*(*4*) **by** *simp*
  next
    case *4*
    then have *cmd-simulog G M j rs* [!] $i =$ *?sas* [!] *0*
      using *cmd-simulog-def False that* **by** *simp*
    then show *?thesis*
      using *4 assms inbounds turing-machine-def that lensas turing-commandD*(*3*)
      by (*metis One-nat-def Suc-1 length-Cons list.size*(*3*) *nth-Cons-0 nth-mem zero-less-numeral*)
  next
    case *5*
    then have $*$: *cmd-simulog G M j rs* [!] $i =$ *?sas* [!] *1*
      using *cmd-simulog-def False that* **by** *simp*
    have *turing-command 2* (*length M*) *G ?cmd*
      using *assms*(*1*) *inbounds turing-machine-def* **by** *simp*
    moreover have *symbols-lt G ?gs*
      using *False less-2-cases-iff numeral-2-eq-2* **by** *simp*
    ultimately have *?sas* [.] $1 < G$
      using *turing-commandD*(*2*) **by** *simp*
    then show *?thesis*
      using *assms* $*$ **by** *simp*

```
    next
      case 6
      then have cmd-simulog G M j rs [!] i = (rs ! i, Stay)
        using cmd-simulog-def False that(3) by simp
      then show ?thesis
        using that by simp
    qed
  qed
  show cmd-simulog G M j rs [.] 0 = rs ! 0 if length rs = k and 0 < k for rs
  proof (cases rs ! (j + 6) ≥ length M ∨ rs ! (j + 7) ≥ G ∨ rs ! (j + 8) ≥ G)
    case True
    then show ?thesis
      using assms that cmd-simulog-def by simp
  next
    case False
    then show ?thesis
      using assms that cmd-simulog-def by simp
  qed
  show ⋀gs. length gs = k ⟹ [*] (cmd-simulog G M j gs) ≤ 1
    using cmd-simulog-def by simp
qed
```

The logging Turing machine consists only of the logging command.

```
definition tm-simulog :: nat ⇒ machine ⇒ tapeidx ⇒ machine where
  tm-simulog G M j ≡ [cmd-simulog G M j]

lemma tm-simulog-tm:
  fixes H :: nat
  assumes turing-machine 2 G M and k ≥ j + 9 and j > 0 and H ≥ Suc (length M) and H ≥ G
  shows turing-machine k H (tm-simulog G M j)
  using turing-command-cmd-simulog tm-simulog-def assms turing-machine-def by simp

lemma transforms-tm-simulogI [transforms-intros]:
  fixes G :: nat and M :: machine and j :: tapeidx
  fixes k :: nat and tps tps' :: tape list and xs :: symbol list
  assumes turing-machine 2 G M and k ≥ j + 9 and j > 0
    and symbols-lt G xs
    and cfg = execute M (start-config 2 xs) t and fst cfg < length M
    and length tps = k
  assumes
    tps ! j = ⌈dummy1⌉
    tps ! (j + 3) = ⌈dummy2⌉
    tps ! (j + 6) = ⌈fst cfg⌉
    tps ! (j + 7) = cfg <!> 0
    tps ! (j + 8) = cfg <!> 1
  assumes tps' = tps
     [j := ⌈direction-to-symbol ((M ! (fst cfg)) (config-read cfg) [~] 0)⌉,
      j + 3 := ⌈direction-to-symbol ((M ! (fst cfg)) (config-read cfg) [~] 1)⌉,
      j + 6 := ⌈fst (execute M (start-config 2 xs) (Suc t))⌉,
      j + 7 := execute M (start-config 2 xs) (Suc t) <!> 0,
      j + 8 := execute M (start-config 2 xs) (Suc t) <!> 1]
  shows transforms (tm-simulog G M j) tps 1 tps'
proof -
  have sem (cmd-simulog G M j) (0, tps) = (1, tps')
  proof (rule semI)
    define H where H = max G (Suc (length M))
    then have H ≥ Suc (length M) H ≥ G
      by simp-all
    then show proper-command k (cmd-simulog G M j)
      using assms cmd-simulog-def by simp
    show length tps = k and length tps' = k
      using assms by simp-all
    show fst (cmd-simulog G M j (read tps)) = 1
```

using *cmd-simulog-def sem'* **by** *simp*

define *rs* **where** *rs = read tps*
then have *lenrs*: *length rs = k*
by (*simp add*: *assms rs-def read-length*)
have *rs6*: *rs ! (j + 6) = fst cfg*
using *rs-def tapes-at-read'[of j + 6 tps] assms* **by** *simp*
then have *inbounds*: *rs ! (j + 6) < length M*
using *assms* **by** *simp*
have *rs7*: *rs ! (j + 7) = cfg <.> 0*
using *rs-def tapes-at-read'[of j + 7 tps] assms* **by** *simp*
then have *rs7-less*: *rs ! (j + 7) < G*
using *assms tape-alphabet[OF assms(1,4)]* **by** *simp*
have *rs8*: *rs ! (j + 8) = cfg <.> 1*
using *rs-def tapes-at-read'[of j + 8 tps] assms* **by** *simp*
then have *rs8-less*: *rs ! (j + 8) < G*
using *assms tape-alphabet[OF assms(1,4)]* **by** *simp*
let *?gs = [rs ! (j + 7), rs ! (j + 8)]*
have *gs*: *?gs = config-read cfg*
proof (*rule nth-equalityI*)
show *length [rs ! (j + 7), rs ! (j + 8)] = length (config-read cfg)*
using *assms execute-num-tapes start-config-length read-length* **by** *simp*
then show *[rs ! (j + 7), rs ! (j + 8)] ! i = config-read cfg ! i*
if *i < length [rs ! (j + 7), rs ! (j + 8)]* **for** *i*
using *assms that rs7 rs8 read-length tapes-at-read'*
by (*metis One-nat-def length-Cons less-2-cases-iff list.size(3) nth-Cons-0 nth-Cons-Suc numeral-2-eq-2*)
qed
let *?cmd = M ! (rs ! (j + 6))*
let *?sas = ?cmd ?gs*
have *lensas*: *length (snd ?sas) = 2*
using *assms(1) inbounds turing-commandD(1) turing-machine-def*
by (*metis One-nat-def Suc-1 canrepr-1 list.size(4) nlength-1-simp nth-mem plus-1-eq-Suc*)
have *sas*: *?sas = (M ! (fst cfg)) (config-read cfg)*
using *rs6 gs* **by** *simp*
have *act (cmd-simulog G M j rs [!] i) (tps ! i) = tps' ! i* **if** *i < k* **for** *i*
proof −
have *cmd-simulog G M j rs =*
(1, map (λi. let sas = (M ! (rs ! (j + 6))) [rs ! (j + 7), rs ! (j + 8)] in
if i = j then (direction-to-symbol (sas [~] 0), Stay)
else if i = j + 3 then (direction-to-symbol (sas [~] 1), Stay)
else if i = j + 6 then (fst sas, Stay)
else if i = j + 7 then sas [!] 0
else if i = j + 8 then sas [!] 1
else (rs ! i, Stay))
[0..<length rs])
using *inbounds rs7-less rs8-less assms cmd-simulog-def* **by** *simp*
then have ∗: *cmd-simulog G M j rs [!] i =*
(if i = j then (direction-to-symbol (?sas [~] 0), Stay)
else if i = j + 3 then (direction-to-symbol (?sas [~] 1), Stay)
else if i = j + 6 then (fst ?sas, Stay)
else if i = j + 7 then ?sas [!] 0
else if i = j + 8 then ?sas [!] 1
else (rs ! i, Stay))
using *that lenrs* **by** *simp*
consider
i = j
| *i = j + 3*
| *i = j + 6*
| *i = j + 7*
| *i = j + 8*
| *i ≠ j ∧ i ≠ j + 3 ∧ i ≠ j + 6 ∧ i ≠ j + 7 ∧ i ≠ j + 8*
by *linarith*
then show *?thesis*

proof (*cases*)
  case *1*
  then have *cmd-simulog G M j rs* [!] *i* = (*direction-to-symbol* (*?sas* [~] *0*), *Stay*)
    using ∗ **by** *simp*
  moreover have *tps'* ! *i* = ⌈*direction-to-symbol* (*?sas* [~] *0*)⌉
    using *1 assms sas* **by** *simp*
  ultimately show *?thesis*
    using *1 act-onesie assms*(*8*) **by** *simp*
next
  case *2*
  then have *cmd-simulog G M j rs* [!] *i* = (*direction-to-symbol* (*?sas* [~] *1*), *Stay*)
    using ∗ **by** *simp*
  moreover have *tps'* ! *i* = ⌈*direction-to-symbol* (*?sas* [~] *1*)⌉
    using *2 assms sas* **by** *simp*
  ultimately show *?thesis*
    using *2 act-onesie assms* **by** *simp*
next
  case *3*
  then have *cmd-simulog G M j rs* [!] *i* = (*fst ?sas*, *Stay*)
    using ∗ **by** *simp*
  moreover have *tps'* ! *i* = ⌈*fst* (*execute M* (*start-config 2 xs*) (*Suc t*))⌉
    using *3 assms* **by** *simp*
  ultimately show *?thesis*
    using *3 act-onesie assms* **by** (*metis exe-lt-length execute.simps*(*2*) *sas sem-fst*)
next
  case *4*
  then have *cmd-simulog G M j rs* [!] *i* = *?sas* [!] *0*
    using ∗ **by** *simp*
  moreover have *tps'* ! *i* = *execute M* (*start-config 2 xs*) (*Suc t*) <!> *0*
    using *4 assms* **by** *simp*
  moreover have *proper-command 2* (*M* ! (*rs* ! (*j* + *6*)))
    using *assms*(*1*,*6*) *rs6 turing-machine-def turing-commandD*(*1*) *turing-machineD* **by** *metis*
  ultimately show *?thesis*
    using *4 assms*(*1*,*11*,*5*,*6*) *exe-lt-length gs read-length rs6 sem-snd turing-machine-def*
    by (*metis execute.simps*(*2*) *length-Cons list.size*(*3*) *numeral-2-eq-2 zero-less-numeral*)
next
  case *5*
  then have *cmd-simulog G M j rs* [!] *i* = *?sas* [!] *1*
    using ∗ **by** *simp*
  moreover have *tps'* ! *i* = *execute M* (*start-config 2 xs*) (*Suc t*) <!> *1*
    using *5 assms* **by** *simp*
  moreover have *proper-command 2* (*M* ! (*rs* ! (*j* + *6*)))
    using *assms*(*1*,*6*) *rs6 turing-machineD turing-commandD*(*1*) **by** *metis*
  ultimately show *?thesis*
    using *5 assms*(*1*,*12*,*5*,*6*) *exe-lt-length gs read-length rs6 sem-snd turing-machine-def*
    by (*metis One-nat-def execute.simps*(*2*) *length-Cons less-2-cases-iff list.size*(*3*) *numeral-2-eq-2*)
next
  case *6*
  then have *cmd-simulog G M j rs* [!] *i* = (*rs* ! *i*, *Stay*)
    using ∗ **by** *simp*
  moreover have *tps'* ! *i* = *tps* ! *i*
    using *6 assms*(*13*) **by** *simp*
  ultimately show *?thesis*
    using *6 assms act-Stay rs-def that* **by** *metis*
qed
qed
then show *act* (*cmd-simulog G M j* (*read tps*) [!] *i*) (*tps* ! *i*) = *tps'* ! *i* **if** *i* < *k* **for** *i*
  using *that rs-def* **by** *simp*
qed
moreover have *execute* (*tm-simulog G M j*) (*0*, *tps*) *1* = *sem* (*cmd-simulog G M j*) (*0*, *tps*)
  using *tm-simulog-def* **by** (*simp add*: *exe-lt-length*)
ultimately have *execute* (*tm-simulog G M j*) (*0*, *tps*) *1* = (*1*, *tps'*)
  by *simp*

then show *?thesis*
using *transforms-def transits-def tm-simulog-def* **by** *auto*
qed

7.3.2 Adjusting head position counters

The Turing machine *tm-simulog* logs the head movements, but what we need is a list of all the head positions during the execution of M. The next TM maintains a number for a head position and adjusts it based on a head movement symbol as provided by *tm-simulog*.

More precisely, the next Turing machine accepts on tape j a symbol encoding a direction, on tape $j+1$ a number representing a head position, and on tape $j+2$ a list of numbers. Depending on the symbol on tape j it decreases, increases or leaves unchanged the number on tape $j+1$. Then it appends this adjusted number to the list on tape $j+2$.

definition *tm-adjust-headpos* :: *nat* ⇒ *machine* **where**
tm-adjust-headpos j ≡
IF λ*rs. rs* ! *j* = □ *THEN*
tm-decr (*j* + *1*)
ELSE
IF λ*rs. rs* ! *j* = **0** *THEN*
tm-incr (*j* + *1*)
ELSE
[]
ENDIF
ENDIF ;;
tm-append (*j* + *2*) (*j* + *1*)

lemma *tm-adjust-headpos-tm*:
assumes *G* ≥ *5* **and** *j* + *2* < *k*
shows *turing-machine k G* (*tm-adjust-headpos j*)
unfolding *tm-adjust-headpos-def*
using *assms turing-machine-branch-turing-machine tm-decr-tm tm-incr-tm tm-append-tm Nil-tm turing-machine-sequential-turing*
by *simp*

locale *turing-machine-adjust-headpos* =
fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *IF* λ*rs. rs* ! *j* = **0** *THEN tm-incr* (*j* + *1*) *ELSE* [] *ENDIF*
definition *tm2* ≡ *IF* λ*rs. rs* ! *j* = □ *THEN tm-decr* (*j* + *1*) *ELSE tm1 ENDIF*
definition *tm3* ≡ *tm2* ;; *tm-append* (*j* + *2*) (*j* + *1*)

lemma *tm3-eq-tm-adjust-headpos*: *tm3* = *tm-adjust-headpos j*
unfolding *tm1-def tm2-def tm3-def tm-adjust-headpos-def* **by** *simp*

context
fixes *tps* :: *tape list* **and** *jj* :: *tapeidx* **and** *k t* :: *nat* **and** *xs* :: *symbol list*
fixes *M* :: *machine*
fixes *G cfg*
assumes *jk*: *length tps* = *k k* ≥ *j* + *3 jj* < *2*
assumes *M*: *turing-machine 2 G M*
assumes *xs*: *symbols-lt G xs*
assumes *cfg*: *cfg* = *execute M* (*start-config 2 xs*) *t fst cfg* < *length M*
assumes *tps0*:
tps ! *j* = ⌈*direction-to-symbol* ((*M* ! (*fst cfg*)) (*config-read cfg*) [~] *jj*)⌉
tps ! (*j* + *1*) = (⌊*cfg* <#> *jj*⌋$_N$, *1*)
tps ! (*j* + *2*) = *nltape* (*map* (λ*t.* (*execute M* (*start-config 2 xs*) *t* <#> *jj*)) [*0*..<*Suc t*])
begin

lemma *k-ge-2*: *2* ≤ *k*
using *jk* **by** *simp*

abbreviation *exc* :: *symbol list* ⇒ *nat* ⇒ *config* **where**

```
  exc y tt ≡ execute M (start-config 2 y) tt

lemma read-tps-j: read tps ! j = direction-to-symbol ((M ! (fst cfg)) (config-read cfg) [~] jj)
  using tps0 onesie-read jk tapes-at-read'
  by (metis less-add-same-cancel1 less-le-trans zero-less-numeral)

lemma write-symbol:
  ∃v. execute M (start-config 2 xs) (Suc t) <!> jj = act (v, (M ! (fst cfg)) (config-read cfg) [~] jj) (cfg <!> jj)
proof −
  let ?d = (M ! (fst cfg)) (config-read cfg) [~] jj
  obtain v where v: (M ! (fst cfg)) (config-read cfg) [!] jj = (v, ?d)
    by (simp add: prod-eq-iff)
  have execute M (start-config 2 xs) (Suc t) <!> jj = exe M cfg <!> jj
    using cfg(1) by simp
  also have ... = sem (M ! (fst cfg)) cfg <!> jj
    by (simp add: cfg(2) exe-lt-length)
  also have ... = act ((M ! (fst cfg)) (config-read cfg) [!] jj) (cfg <!> jj)
    using sem-snd-tm M cfg execute-num-tapes start-config-length jk
    by (metis (no-types, lifting) numeral-2-eq-2 prod.exhaust-sel zero-less-Suc)
  also have ... = act (v, ?d) (cfg <!> jj)
    using v by simp
  finally have *: execute M (start-config 2 xs) (Suc t) <!> jj = act (v, ?d) (cfg <!> jj) .
  then show ?thesis
    by auto
qed

lemma tm1 [transforms-intros]:
  assumes ttt = 7 + 2 * nlength (cfg <#> jj)
    and (M ! (fst cfg)) (config-read cfg) [~] jj ≠ Left
    and tps' = tps[j + 1 := (⌊execute M (start-config 2 xs) (Suc t) <#> jj⌋N, 1)]
  shows transforms tm1 tps ttt tps'
  unfolding tm1-def
proof (tform)
  let ?d = (M ! (fst cfg)) (config-read cfg) [~] jj
  obtain v where v: execute M (start-config 2 xs) (Suc t) <!> jj = act (v, ?d) (cfg <!> jj)
    using write-symbol by auto

  { assume read tps ! j = 2
    then have ?d = Right
      using read-tps-j assms(2) direction-to-symbol-def by (cases ?d) simp-all
    show j + 1 < length tps
      using jk by simp
    show tps ! (j + 1) = (⌊cfg <#> jj⌋N, 1)
      using tps0 by simp-all
    show tps' = tps[j + 1 := (⌊Suc (cfg <#> jj)⌋N, 1)]
    proof −
      have execute M (start-config 2 xs) (Suc t) <!> jj = cfg <!> jj |:=| v |+| 1
        using v ‹?d = Right› act-Right' by simp
      then have execute M (start-config 2 xs) (Suc t) <#> jj = cfg <#> jj + 1
        by simp
      then show ?thesis
        using assms(3) by simp
    qed
  }
  { assume read tps ! j ≠ 2
    then have ?d = Stay
      using read-tps-j assms(2) direction-to-symbol-def by (cases ?d) simp-all
    then have execute M (start-config 2 xs) (Suc t) <!> jj = cfg <!> jj |:=| v
      using v act-Stay' by simp
    then have execute M (start-config 2 xs) (Suc t) <#> jj = cfg <#> jj
      by simp
    then show tps' = tps
      using assms(3) tps0 by (metis list-update-id)
```

```
  }
  show (5 + 2 ∗ nlength (cfg <#> jj)) + 2 ≤ ttt 0 + 1 ≤ ttt
    using assms(1) by simp-all
qed

lemma tm2 [transforms-intros]:
  assumes ttt = 10 + 2 ∗ nlength (cfg <#> jj)
    and tps′ = tps[j + 1 := (⌊execute M (start-config 2 xs) (Suc t) <#> jj⌋N, 1)]
  shows transforms tm2 tps ttt tps′
  unfolding tm2-def
proof (tform tps: k-ge-2 jk assms)
  let ?d = (M ! (fst cfg)) (config-read cfg) [~] jj
  show 8 + 2 ∗ nlength (cfg <#> jj) + 2 ≤ ttt
    using assms(1) by simp
  show read tps ! j ≠ □ ⟹ ?d ≠ Left
    using read-tps-j direction-to-symbol-def by (cases ?d) simp-all
  { assume 0: read tps ! j = □
    show tps ! (j + 1) = (⌊cfg <#> jj⌋N, 1)
      using tps0 by simp
    show tps′ = tps[j + 1 := (⌊cfg <#> jj − 1⌋N, 1)]
    proof −
      let ?d = (M ! (fst cfg)) (config-read cfg) [~] jj
      obtain v where v: execute M (start-config 2 xs) (Suc t) <!> jj = act (v, ?d) (cfg <!> jj)
        using write-symbol by auto
      then have ?d = Left
        using 0 read-tps-j assms(2) direction-to-symbol-def by (cases ?d) simp-all
      then have execute M (start-config 2 xs) (Suc t) <!> jj = cfg <!> jj |:=| v |−| 1
        using v act-Left′ by simp
      then have execute M (start-config 2 xs) (Suc t) <#> jj = cfg <#> jj − 1
        by simp
      then show ?thesis
        using assms(2) by simp
    qed
  }
qed

lemma tm3:
  assumes ttt = 16 + 2 ∗ nlength (cfg <#> jj) + 2 ∗ nlength (exc xs (Suc t) <#> jj)
    and tps′ = tps
      [j + 1 := (⌊execute M (start-config 2 xs) (Suc t) <#> jj⌋N, 1),
       j + 2 := nltape (map (λt. (execute M (start-config 2 xs) t <#> jj)) [0..<Suc (Suc t)])]
  shows transforms tm3 tps ttt tps′
  unfolding tm3-def
proof (tform tps: jk assms)
  let ?ns = (map (λt. (execute M (start-config 2 xs) t <#> jj)) [0..<Suc t])
  let ?i = Suc (nllength (map (λt. (execute M (start-config 2 xs) t <#> jj)) [0..<Suc t]))
  let ?n = exc xs (Suc t) <#> jj
  let ?tps = tps[j + 1 := (⌊exc xs (Suc t) <#> jj⌋N, 1)]
  show ?tps ! (j + 2) = (⌊?ns⌋NL, ?i)
    using tps0 by simp
  show Suc (nllength ?ns) ≤ ?i
    by simp
  show tps′ = tps
    [j + 1 := (⌊?n⌋N, 1),
     j + 2 := nltape (?ns @ [snd (exe M (exc xs t)) :#: jj])]
    using assms(2) nlcontents-def nllength-def by simp
  show ttt =
      10 + 2 ∗ nlength (cfg <#> jj) +
      (7 + nllength (map (λt. exc xs t <#> jj) [0..<Suc t]) −
       Suc (nllength (map (λt. exc xs t <#> jj) [0..<Suc t])) +
       2 ∗ nlength (snd (exe M (exc xs t)) :#: jj))
    using assms(1) by simp
qed
```

```
end

end

lemma transforms-tm-adjust-headposI [transforms-intros]:
  fixes j :: tapeidx
  fixes tps tps' :: tape list and k jj t :: nat and xs :: symbol list
    and M :: machine and G :: nat and cfg :: config
  assumes turing-machine 2 G M
    and length tps = k and k ≥ j + 3 and jj < 2
    and symbols-lt G xs
    and cfg: cfg = execute M (start-config 2 xs) t fst cfg < length M
  assumes
    tps ! j = ⌈direction-to-symbol ((M ! (fst cfg)) (config-read cfg) [~] jj)⌉
    tps ! (j + 1) = (⌊cfg <#> jj⌋_N, 1)
    tps ! (j + 2) = nltape (map (λt. (execute M (start-config 2 xs) t <#> jj)) [0..<Suc t])
  assumes max-head-pos: ∀t. execute M (start-config 2 xs) t <#> jj ≤ max-head-pos
  assumes ttt: ttt = 16 + 4 ∗ nlength max-head-pos
  assumes tps' = tps
    [j + 1 := (⌊execute M (start-config 2 xs) (Suc t) <#> jj⌋_N, 1),
     j + 2 := nltape (map (λt. (execute M (start-config 2 xs) t <#> jj)) [0..<Suc (Suc t)])]
  shows transforms (tm-adjust-headpos j) tps ttt tps'
proof −
  interpret loc: turing-machine-adjust-headpos j .
  let ?ttt = 16 + 2 ∗ nlength (cfg <#> jj) + 2 ∗ nlength (execute M (start-config 2 xs) (Suc t) <#> jj)
  have transforms (tm-adjust-headpos j) tps ?ttt tps'
    using assms loc.tm3-eq-tm-adjust-headpos loc.tm3 by simp
  moreover have ?ttt ≤ ttt
  proof −
    have ?ttt ≤ 16 + 2 ∗ nlength (cfg <#> jj) + 2 ∗ nlength max-head-pos
      using max-head-pos nlength-mono by (meson add-le-mono le-refl mult-le-mono2)
    also have ... ≤ 16 + 2 ∗ nlength max-head-pos + 2 ∗ nlength max-head-pos
      using max-head-pos cfg nlength-mono by simp
    also have ... = 16 + 4 ∗ nlength max-head-pos
      by simp
    finally show ?thesis
      using ttt by simp
  qed
  ultimately show ?thesis
    using transforms-monotone by simp
qed
```

7.3.3 Listing the head positions

The next Turing machine is essentially a loop around the TM *tm-simulog*, which outputs head movements, combined with two instances of the TM *tm-adjust-headpos*, each of which maintains a head positions lists. The loop ends when the simulated machine reaches the halting state. If the simulated machine does not halt, neither does the simulator, but we will not consider this case when we analyze the semantics. The TM receives an input on tape $j + 7$. During the simulation of M this tape is a replica of the simulated machine's input tape, and tape $j + 8$ is a replica of the work/output tape. The lists of the head positions will be on tapes $j + 2$ and $j + 5$ for the input tape and work/output tape, respectively.

```
definition tm-list-headpos :: nat ⇒ machine ⇒ tapeidx ⇒ machine where
  tm-list-headpos G M j ≡
    tm-right-many {j + 1, j + 2, j + 4, j+ 5} ;;
    tm-write (j + 6) □ ;;
    tm-append (j + 2) (j + 1) ;;
    tm-append (j + 5) (j + 4) ;;
    WHILE [] ; λrs. rs ! (j + 6) < length M DO
      tm-simulog G M j ;;
      tm-adjust-headpos j ;;
      tm-adjust-headpos (j + 3) ;;
```

tm-write-many $\{j, j + 3\}$ ▷
DONE ;;
tm-write $(j + 6)$ ▷ ;;
tm-cr $(j + 2)$;;
tm-cr $(j + 5)$

lemma *tm-list-headpos-tm*:
fixes H :: *nat*
assumes *turing-machine 2 G M* **and** $k \geq j + 9$ **and** $j > 0$ **and** $H \geq Suc\ (length\ M)$ **and** $H \geq G$
assumes $H \geq 5$
shows *turing-machine k H (tm-list-headpos G M j)*
unfolding *tm-list-headpos-def*
using *assms turing-machine-loop-turing-machine turing-machine-sequential-turing-machine Nil-tm tm-append-tm tm-simulog-tm tm-adjust-headpos-tm tm-right-many-tm tm-write-tm tm-write-many-tm tm-cr-tm*
by *simp*

locale *turing-machine-list-headpos* =
fixes G :: *nat* **and** M :: *machine* **and** j :: *tapeidx*
begin

definition *tm1* ≡ *tm-right-many* $\{j + 1, j + 2, j + 4, j + 5\}$
definition *tm2* ≡ *tm1* ;; *tm-write* $(j + 6)$ □
definition *tm3* ≡ *tm2* ;; *tm-append* $(j + 2)$ $(j + 1)$
definition *tm4* ≡ *tm3* ;; *tm-append* $(j + 5)$ $(j + 4)$
definition *tmL1* ≡ *tm-simulog G M j*
definition *tmL2* ≡ *tmL1* ;; *tm-adjust-headpos j*
definition *tmL3* ≡ *tmL2* ;; *tm-adjust-headpos* $(j + 3)$
definition *tmL4* ≡ *tmL3* ;; *tm-write-many* $\{j, j + 3\}$ ▷
definition *tmL* ≡ *WHILE* [] ; $\lambda rs.\ rs\ !\ (j + 6) < length\ M$ *DO tmL4 DONE*
definition *tm5* ≡ *tm4* ;; *tmL*
definition *tm6* ≡ *tm5* ;; *tm-write* $(j + 6)$ ▷
definition *tm7* ≡ *tm6* ;; *tm-cr* $(j + 2)$
definition *tm8* ≡ *tm7* ;; *tm-cr* $(j + 5)$

lemma *tm8-eq-tm-list-headpos*: *tm8* = *tm-list-headpos G M j*
unfolding *tm1-def tm2-def tm3-def tm4-def tmL1-def tmL2-def tmL3-def tmL4-def tmL-def tm5-def tm6-def tm7-def tm8-def*
tm-list-headpos-def
by *simp*

context
fixes *tps0* :: *tape list*
fixes *thalt k* :: *nat* **and** *xs* :: *symbol list*
assumes *M*: *turing-machine 2 G M*
assumes *jk*: $k \geq j + 9$ $j > 0$ $length\ tps0 = k$
assumes *thalt*:
$\forall t<thalt.\ fst\ (execute\ M\ (start\text{-}config\ 2\ xs)\ t) < length\ M$
$fst\ (execute\ M\ (start\text{-}config\ 2\ xs)\ thalt) = length\ M$
assumes *xs*: *symbols-lt G xs*
assumes *tps0*:
$tps0\ !\ j = \lceil \triangleright \rceil$
$tps0\ !\ (j + 1) = (\lfloor 0 \rfloor_N,\ 0)$
$tps0\ !\ (j + 2) = (\lfloor [] \rfloor_{NL},\ 0)$
$tps0\ !\ (j + 3) = \lceil \triangleright \rceil$
$tps0\ !\ (j + 4) = (\lfloor 0 \rfloor_N,\ 0)$
$tps0\ !\ (j + 5) = (\lfloor [] \rfloor_{NL},\ 0)$
$tps0\ !\ (j + 6) = \lceil \triangleright \rceil$
$tps0\ !\ (j + 7) = (\lfloor xs \rfloor,\ 0)$
$tps0\ !\ (j + 8) = (\lfloor [] \rfloor,\ 0)$
begin

abbreviation *exec* :: $nat \Rightarrow config$ **where**
exec t ≡ *execute M (start-config 2 xs) t*

lemma *max-head-pos-0*: ∀ *t*. *exec t* <#> *0* ≤ *thalt*
  using *thalt M head-pos-le-halting-time* **by** *simp*

lemma *max-head-pos-1*: ∀ *t*. *exec t* <#> *1* ≤ *thalt*
  using *thalt M head-pos-le-halting-time* **by** *simp*

definition *tps1* ≡ *tps0*
  [(*j* + *1*) := (⌊*0*⌋$_N$, *1*),
   (*j* + *2*) := (⌊[]⌋$_{NL}$, *1*),
   (*j* + *4*) := (⌊*0*⌋$_N$, *1*),
   (*j* + *5*) := (⌊[]⌋$_{NL}$, *1*),
   (*j* + *6*) := ⌈▷⌉]

lemma *tm1* [*transforms-intros*]:
  assumes *ttt* = *1*
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def*
proof (*tform tps*: *assms tps0 jk tps1-def*)
  show *tps1* = *map* (λ*i*. *if i* ∈ {*j* + *1*, *j* + *2*, *j* + *4*, *j* + *5*} *then tps0* ! *i* |+| *1*
                  else tps0 ! *i*) [*0*..<*length tps0*]
    (**is** *tps1* = *?rhs*)
  proof (*rule nth-equalityI*)
    show *len*: *length tps1* = *length ?rhs*
      by (*simp add*: *tps1-def*)
    let *?J* = {*j* + *1*, *j* + *2*, *j* + *4*, *j* + *5*}
    show *tps1* ! *i* = *?rhs* ! *i* **if** *i* < *length tps1* **for** *i*
    proof (*cases i* ∈ *?J*)
      case *True*
      have *tps1* ! (*j* + *1*) = *?rhs* ! (*j* + *1*)
        using *tps1-def jk tps0* **by** *fastforce*
      moreover have *tps1* ! (*j* + *2*) = *?rhs* ! (*j* + *2*)
        using *tps1-def jk tps0* **by** *fastforce*
      moreover have *tps1* ! (*j* + *4*) = *?rhs* ! (*j* + *4*)
        using *tps1-def jk tps0* **by** *fastforce*
      moreover have *tps1* ! (*j* + *5*) = *?rhs* ! (*j* + *5*)
        using *tps1-def jk tps0* **by** *fastforce*
      ultimately show *?thesis*
        using *True* **by** *fast*
    next
      case *notinJ*: *False*
      then have ∗: *?rhs* ! *i* = *tps0* ! *i*
        using *that len* **by** *simp*
      show *?thesis*
      proof (*cases i* = *j* + *6*)
        case *True*
        then show *?thesis*
          using ∗ *that tps0*(*7*) *tps1-def* **by** *simp*
      next
        case *False*
        then have *tps1* ! *i* = *tps0* ! *i*
          using *tps1-def notinJ that* **by** *simp*
        then show *?thesis*
          using ∗ **by** *simp*
      qed
    qed
  qed
qed

definition *tps2* ≡ *tps0*
  [(*j* + *1*) := (⌊*0*⌋$_N$, *1*),
   (*j* + *2*) := (⌊[]⌋$_{NL}$, *1*),
   (*j* + *4*) := (⌊*0*⌋$_N$, *1*),

$(j + 5) := (\lfloor[]\rfloor_{NL}, 1),$
$(j + 6) := \lceil\square\rceil]$

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *2*
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def*
proof (*tform tps*: *assms tps0 jk tps2-def tps1-def*)
show *tps2* = *tps1*[*j* + *6* := *tps1* ! (*j* + *6*) |:=| *0*]
using *tps2-def tps1-def jk onesie-write*
by (*smt* (*verit*) *list-update-beyond list-update-overwrite nth-list-update-eq verit-comp-simplify1*(*3*))
qed

definition *tps3* ≡ *tps0*
$[(j + 1) := (\lfloor 0\rfloor_N, 1),$
$(j + 2) := nltape\ [0],$
$(j + 4) := (\lfloor 0\rfloor_N, 1),$
$(j + 5) := (\lfloor[]\rfloor_{NL}, 1),$
$(j + 6) := \lceil\square\rceil]$

lemma *tm3* [*transforms-intros*]:
assumes *ttt* = *8*
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def*
proof (*tform tps*: *tps0 jk tps2-def tps3-def*)
show *tps3* = *tps2*[*j* + *2* := *nltape* ([] @ [*0*])]
using *tps3-def jk tps2-def*
by (*smt* (*verit*, *ccfv-SIG*) *Suc3-eq-add-3 add-2-eq-Suc add-less-cancel-left lessI list-update-overwrite list-update-swap not-add-less2 numeral-2-eq-2 numeral-3-eq-3 numeral-Bit0 numeral-plus-numeral self-append-conv2 semiring-norm*(*4*) *semiring-norm*(*5*))
show *ttt* = *2* + (*7* + *nllength* [] − *Suc 0* + *2* ∗ *nlength 0*)
using *assms* **by** *simp*
qed

definition *tps4* ≡ *tps0*
$[(j + 1) := (\lfloor 0\rfloor_N, 1),$
$(j + 2) := nltape\ [0],$
$(j + 4) := (\lfloor 0\rfloor_N, 1),$
$(j + 5) := nltape\ [0],$
$(j + 6) := \lceil\square\rceil]$

lemma *tm4* [*transforms-intros*]:
assumes *ttt* = *14*
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform tps*: *tps3-def jk tps0 tps4-def*)
show *tps4* = *tps3*[*j* + *5* := *nltape* ([] @ [*0*])]
unfolding *tps3-def tps4-def*
using *jk tps0*
by (*smt* (*verit*, *ccfv-threshold*) *add-Suc add-Suc-right append.left-neutral eval-nat-numeral*(*3*) *list-update-overwrite list-update-swap n-not-Suc-n nlcontents-Nil numeral-Bit0*)
show *ttt* = *8* + (*7* + *nllength* [] − *Suc 0* + *2* ∗ *nlength 0*)
using *assms* **by** *simp*
qed

The tapes after t iterations:

definition *tpsL t* ≡ *tps0*
$[(j + 1) := (\lfloor exec\ t\ \text{<\#>}\ 0\rfloor_N, 1),$
$(j + 2) := nltape\ (map\ (\lambda t.\ exec\ t\ \text{<\#>}\ 0)\ [0..{<}Suc\ t]),$
$(j + 4) := (\lfloor exec\ t\ \text{<\#>}\ 1\rfloor_N, 1),$
$(j + 5) := nltape\ (map\ (\lambda t.\ exec\ t\ \text{<\#>}\ 1)\ [0..{<}Suc\ t]),$
$(j + 6) := \lceil fst\ (exec\ t)\rceil,$
$(j + 7) := exec\ t\ \text{<!>}\ 0,$

```
  (j + 8) := exec t <!> 1]

lemma lentpsL: length (tpsL t) = k
  using jk tpsL-def by simp

lemma tpsL-0-eq-tps4: tpsL 0 = tps4
proof -
  have *: exec 0 = (0, [(⌊xs⌋, 0), (⌊[]⌋, 0)])
    using start-config-def contents-def by auto
  show ?thesis
    (is tpsL 0 = ?rhs)
  proof (rule nth-equalityI)
    show length (tpsL 0) = length ?rhs
      by (simp add: tps4-def tpsL-def)
    show tpsL 0 ! i = ?rhs ! i if i < length (tpsL 0) for i
    proof -
      show ?thesis
      proof (cases i = j ∨ i = j + 1 ∨ i = j + 2 ∨ i = j + 4 ∨ i = j + 5 ∨ i = j + 6 ∨ i = j + 7 ∨ i = j +
8)
        case True
        then show ?thesis
          unfolding tps4-def tpsL-def using * tps0 jk by auto
      next
        case False
        then show ?thesis
          unfolding tps4-def tpsL-def using * tps0 jk that by (smt (verit) nth-list-update-neq)
      qed
    qed
  qed
qed

definition tpsL1 t ≡ tps0
  [j := ⌈direction-to-symbol ((M ! fst (exec t)) (config-read (exec t)) [~] 0)⌉,
   j + 1 := (⌊exec t <#> 0⌋_N, 1),
   j + 2 := nltape (map (λt. exec t <#> 0) [0..<Suc t]),
   j + 3 := ⌈direction-to-symbol ((M ! fst (exec t)) (config-read (exec t)) [~] 1)⌉,
   j + 4 := (⌊exec t <#> 1⌋_N, 1),
   j + 5 := nltape (map (λt. exec t <#> 1) [0..<Suc t]),
   j + 6 := ⌈fst (exec (Suc t))⌉,
   j + 7 := exec (Suc t) <!> 0,
   j + 8 := exec (Suc t) <!> 1]

lemma lentpsL1: length (tpsL1 t) = k
    using jk tpsL1-def by (simp only: length-list-update)

lemma tmL1 [transforms-intros]:
  assumes fst (exec t) < length M
  shows transforms tmL1 (tpsL t) 1 (tpsL1 t)
  unfolding tmL1-def
proof (tform tps: M xs jk assms)
  show j + 9 ≤ length (tpsL t)
    using tpsL-def jk by (simp only: length-list-update)
  show tpsL t ! j = ⌈▷⌉
    using tpsL-def tps0 by (simp only: nth-list-update-eq nth-list-update-neq)
  show tpsL t ! (j + 3) = ⌈▷⌉
    using tpsL-def tps0 by (simp only: nth-list-update-eq nth-list-update-neq)
  show tpsL t ! (j + 6) = ⌈fst (exec t)⌉
    using tpsL-def tps0 jk by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  show tpsL t ! (j + 7) = exec t <!> 0
    using tpsL-def tps0 jk by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  show  tpsL t ! (j + 8) = exec t <!> 1
    using tpsL-def tps0 jk by (simp only: length-list-update nth-list-update-eq nth-list-update-neq One-nat-def)
  show tpsL1 t = (tpsL t)
```

[j := ⌈*direction-to-symbol* ((*M* ! *fst* (*exec t*)) (*config-read* (*exec t*)) [~] *0*)⌉,
 j + *3* := ⌈*direction-to-symbol* ((*M* ! *fst* (*exec t*)) (*config-read* (*exec t*)) [~] *1*)⌉,
 j + *6* := ⌈*fst* (*exec* (*Suc t*))⌉,
 j + *7* := *exec* (*Suc t*) <!> *0*,
 j + *8* := *exec* (*Suc t*) <!> *1*]
unfolding *tpsL1-def tpsL-def*
by (*simp only*: *list-update-overwrite list-update-swap-less*[*of j+7*] *list-update-swap-less*[*of j+6*]
 list-update-swap-less[*of j+3*] *list-update-swap-less*[*of j*])
qed

definition *tpsL2 t* ≡ *tps0*
[*j* := ⌈*direction-to-symbol* ((*M* ! *fst* (*exec t*)) (*config-read* (*exec t*)) [~] *0*)⌉,
 j + *1* := (⌊*exec* (*Suc t*) <#> *0*⌋$_N$, *1*),
 j + *2* := *nltape* (*map* (λ*t*. *exec t* <#> *0*) [*0*..<*Suc* (*Suc t*)]),
 j + *3* := ⌈*direction-to-symbol* ((*M* ! *fst* (*exec t*)) (*config-read* (*exec t*)) [~] *1*)⌉,
 j + *4* := (⌊*exec t* <#> *1*⌋$_N$, *1*),
 j + *5* := *nltape* (*map* (λ*t*. *exec t* <#> *1*) [*0*..<*Suc t*]),
 j + *6* := ⌈*fst* (*exec* (*Suc t*))⌉,
 j + *7* := *exec* (*Suc t*) <!> *0*,
 j + *8* := *exec* (*Suc t*) <!> *1*]

lemma *lentpsL2*: *length* (*tpsL2 t*) = *k*
 using *jk tpsL2-def* **by** (*simp only*: *length-list-update*)

lemma *tmL2* [*transforms-intros*]:
 assumes *fst* (*exec t*) < *length M*
 and *ttt* = *17* + *4* ∗ *nlength thalt*
 shows *transforms tmL2* (*tpsL t*) *ttt* (*tpsL2 t*)
 unfolding *tmL2-def*
proof (*tform tps*: *M xs jk assms*(*1*))
 show ∀ *t*. *exec t* <#> *0* ≤ *thalt*
 using *max-head-pos-0* **by** *simp*
 show *j* + *3* ≤ *length* (*tpsL1 t*)
 using *lentpsL1 jk* **by** *simp*
 show (*0* :: *nat*) < *2*
 by *simp*
 show *tpsL1 t* ! *j* = ⌈*direction-to-symbol* ((*M* ! *fst* (*exec t*)) (*config-read* (*exec t*)) [~] *0*)⌉
 using *tpsL1-def jk lentpsL1* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
 show *tpsL1 t* ! (*j* + *1*) = (⌊*exec t* <#> *0*⌋$_N$, *1*)
 using *tpsL1-def jk* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
 show *tpsL1 t* ! (*j* + *2*) = *nltape* (*map* (λ*t*. *snd* (*exec t*) :#: *0*) [*0*..<*Suc t*])
 using *tpsL1-def jk lentpsL1* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
 show *tpsL2 t* = (*tpsL1 t*)
 [*j* + *1* := (⌊*exec* (*Suc t*) <#> *0*⌋$_N$, *1*),
 j + *2* := *nltape* (*map* (λ*t*. *exec t* <#> *0*) [*0*..<*Suc* (*Suc t*)])]
 unfolding *tpsL1-def tpsL2-def*
 by (*simp only*: *list-update-overwrite list-update-swap-less*[*of j+1*] *list-update-swap-less*[*of j+2*])
 show *ttt* = *1* + (*16* + *4* ∗ *nlength thalt*)
 using *assms*(*2*) **by** *simp*
qed

definition *tpsL3 t* ≡ *tps0*
[*j* := ⌈*direction-to-symbol* ((*M* ! *fst* (*exec t*)) (*config-read* (*exec t*)) [~] *0*)⌉,
 j + *1* := (⌊*exec* (*Suc t*) <#> *0*⌋$_N$, *1*),
 j + *2* := *nltape* (*map* (λ*t*. *exec t* <#> *0*) [*0*..<*Suc* (*Suc t*)]),
 j + *3* := ⌈*direction-to-symbol* ((*M* ! *fst* (*exec t*)) (*config-read* (*exec t*)) [~] *1*)⌉,
 j + *4* := (⌊*exec* (*Suc t*) <#> *1*⌋$_N$, *1*),
 j + *5* := *nltape* (*map* (λ*t*. *exec t* <#> *1*) [*0*..<*Suc* (*Suc t*)]),
 j + *6* := ⌈*fst* (*exec* (*Suc t*))⌉,
 j + *7* := *exec* (*Suc t*) <!> *0*,
 j + *8* := *exec* (*Suc t*) <!> *1*]

lemma *lentpsL3*: *length* (*tpsL3 t*) = *k* using *jk tpsL3-def* **by** (*simp only*: *length-list-update*)

lemma *tmL3* [*transforms-intros*]:
assumes *fst* (*exec t*) < *length M* **and** *ttt* = *33* + *8* ∗ *nlength thalt*
shows *transforms tmL3* (*tpsL t*) *ttt* (*tpsL3 t*)
unfolding *tmL3-def*
proof (*tform tps*: *M jk assms*(*1*))
show ∀ *t*. *exec t* <#> *1* ≤ *thalt*
using *max-head-pos-1* **.**
show *j* + *3* + *3* ≤ *length* (*tpsL2 t*)
using *tpsL2-def jk* **by** (*simp only*: *length-list-update*)
show *symbols-lt G xs*
using *xs* **.**
show (*1* :: *nat*) < *2*
by *simp*
show *tpsL2 t* ! (*j* + *3*) = ⌈*direction-to-symbol* ((*M* ! *fst* (*exec t*)) (*config-read* (*exec t*)) [~] *1*)⌉
using *tpsL2-def jk lentpsL2* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
have *j* + *3* + *1* = *j* + *4*
by *simp*
then show *tpsL2 t* ! (*j* + *3* + *1*) = (⌊*exec t* <#> *1*⌋$_N$, *1*)
using *tpsL2-def jk* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
have *j* + *3* + *2* = *j* + *5*
by *simp*
then show *tpsL2 t* ! (*j* + *3* + *2*) = *nltape* (*map* (λ*t*. *exec t* <#> *1*) [*0*..<*Suc t*])
using *tpsL2-def jk lentpsL2* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
have *tpsL3 t* = (*tpsL2 t*)
[*j* + *4* := (⌊*snd* (*exec* (*Suc t*)) :#: *1*⌋$_N$, *1*),
j + *5* := *nltape* (*map* (λ*t*. *snd* (*exec t*) :#: *1*) [*0*..<*Suc* (*Suc t*)])]
unfolding *tpsL2-def tpsL3-def*
by (*simp only*: *list-update-overwrite list-update-swap-less*[*of j*+*4*] *list-update-swap-less*[*of j*+*5*])
moreover have *j* + *3* + *1* = *j* + *4* *j* + *3* + *2* = *j* + *5*
by *simp-all*
ultimately show *tpsL3 t* = (*tpsL2 t*)
[*j* + *3* + *1* := (⌊*snd* (*exec* (*Suc t*)) :#: *1*⌋$_N$, *1*),
j + *3* + *2* := *nltape* (*map* (λ*t*. *snd* (*exec t*) :#: *1*) [*0*..<*Suc* (*Suc t*)])]
by *metis*
show *ttt* = *17* + *4* ∗ *nlength thalt* + (*16* + *4* ∗ *nlength thalt*)
using *assms*(*2*) **by** *simp*
qed

definition *tpsL4 t* ≡ *tps0*
[*j* + *1* := (⌊*exec* (*Suc t*) <#> *0*⌋$_N$, *1*),
j + *2* := *nltape* (*map* (λ*t*. *exec t* <#> *0*) [*0*..<*Suc* (*Suc t*)]),
j + *4* := (⌊*exec* (*Suc t*) <#> *1*⌋$_N$, *1*),
j + *5* := *nltape* (*map* (λ*t*. *exec t* <#> *1*) [*0*..<*Suc* (*Suc t*)]),
j + *6* := ⌈*fst* (*exec* (*Suc t*))⌉,
j + *7* := *exec* (*Suc t*) <!> *0*,
j + *8* := *exec* (*Suc t*) <!> *1*]

lemma *lentpsL4*: *length* (*tpsL4 t*) = *k*
using *jk tpsL4-def* **by** (*simp only*: *length-list-update*)

lemma *tmL4*:
assumes *fst* (*exec t*) < *length M*
and *ttt* = *34* + *8* ∗ *nlength thalt*
shows *transforms tmL4* (*tpsL t*) *ttt* (*tpsL4 t*)
unfolding *tmL4-def*
proof (*tform tps*: *jk assms*(*1*) *lentpsL3 lentpsL4 time*: *assms*)
have *tpsL4 t* ! *i* = *tpsL3 t* ! *i* |:=| *Suc 0* **if** *i* = *j* ∨ *i* = *j* + *3* **for** *i*
proof (*cases i* = *j*)
case *True*
then show *?thesis*
using *tpsL3-def tpsL4-def jk lentpsL4 onesie-write tps0*

by (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq One-nat-def*)
next
case *False*
then have $i = j + 3$
using *that* **by** *simp*
then show *?thesis*
using *tpsL3-def tpsL4-def jk lentpsL4 onesie-write tps0*
by (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq One-nat-def*)
qed
then show $\bigwedge ja.\ ja \in \{j, j + 3\} \Longrightarrow tpsL4\ t\ !\ ja = tpsL3\ t\ !\ ja\ |{:=}|\ 1$
by *simp*
have $tpsL4\ t\ !\ i = tpsL3\ t\ !\ i$ **if** $i < length\ (tpsL4\ t)$ **and** $i \neq j \wedge i \neq j + 3$ **for** i
proof −
consider
$i = j \mid i = j + 1 \mid i = j + 2 \mid i = j + 3 \mid i = j + 4 \mid i = j + 5 \mid i = j + 6 \mid i = j + 7 \mid i = j + 8$
$\mid i < j \mid i > j + 8$
by *linarith*
then show *?thesis*
using *tpsL3-def tpsL4-def that*
by (*cases*) (*auto simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
qed
then show $\bigwedge ja.\ ja < length\ (tpsL4\ t) \Longrightarrow ja \notin \{j, j + 3\} \Longrightarrow tpsL4\ t\ !\ ja = tpsL3\ t\ !\ ja$
by *simp*
qed

lemma *tpsL4-Suc*: $tpsL4\ t = tpsL\ (Suc\ t)$ (**is** $?l = ?r$)
proof (*rule nth-equalityI*)
show $length\ ?l = length\ ?r$
using *lentpsL4 tpsL-def jk* **by** *simp*
show $?l\ !\ i = ?r\ !\ i$ **if** $i < length\ ?l$ **for** i
proof −
consider
$i = j \mid i = j + 1 \mid i = j + 2 \mid i = j + 3 \mid i = j + 4 \mid i = j + 5 \mid i = j + 6 \mid i = j + 7 \mid i = j + 8$
$\mid i < j \mid i > j + 8$
by *linarith*
then show *?thesis*
using *tpsL4-def tpsL-def*
by (*cases*) (*simp-all only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
qed
qed

lemma *tmL4'*:
assumes $fst\ (exec\ t) < length\ M$
and $ttt = 34 + 8 * nlength\ thalt$
shows $transforms\ tmL4\ (tpsL\ t)\ ttt\ (tpsL\ (Suc\ t))$
using *tpsL4-Suc tmL4 assms* **by** *simp*

lemma *tmL*:
assumes $ttt = thalt * (36 + 8 * nlength\ thalt) + 1$
shows $transforms\ tmL\ (tpsL\ 0)\ ttt\ (tpsL\ thalt)$
unfolding *tmL-def*
proof (*tform*)
have $tpsL\ t\ !\ (j + 6) = \lceil fst\ (exec\ t) \rceil$ **for** t
using *tpsL-def jk* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
moreover have $j + 6 < length\ (tpsL\ t)$
using *jk tpsL-def* **by** *simp*
ultimately have $*$: $read\ (tpsL\ t)\ !\ (j + 6) = fst\ (exec\ t)$ **for** t
using *tapes-at-read'*[*of* $j + 6$ *tpsL t*] *onesie-read*[*of fst* (*exec t*)]
by (*simp add*: *lentpsL*)
show $\bigwedge t.\ t < thalt \Longrightarrow read\ (tpsL\ t)\ !\ (j + 6) < length\ M$
using $*$ *thalt* **by** *simp*
show $\bigwedge t.\ t < thalt \Longrightarrow transforms\ tmL4\ (tpsL\ t)\ (34 + 8 * nlength\ thalt)\ (tpsL\ (Suc\ t))$
using *tmL4'* $*$ *thalt*(*1*) **by** *simp*

```
  show ¬ read (tpsL thalt) ! (j + 6) < length M
    using * thalt(2) by simp
  show thalt * tosym (34 + 8 * nlength thalt) + 1 ≤ ttt
    using assms by simp
qed

lemma tmL' [transforms-intros]:
  assumes ttt = thalt * (36 + 8 * nlength thalt) + 1
  shows transforms tmL tps4 ttt (tpsL thalt)
  using assms tmL tpsL-0-eq-tps4 by simp

definition tps5 ≡ tps0
  [(j + 1) := (⌊exec thalt <#> 0⌋N, 1),
   (j + 2) := nltape (map (λt. exec t <#> 0) [0..<Suc thalt]),
   (j + 4) := (⌊exec thalt <#> 1⌋N, 1),
   (j + 5) := nltape (map (λt. exec t <#> 1) [0..<Suc thalt]),
   (j + 6) := ⌈fst (exec thalt)⌉,
   (j + 7) := exec thalt <!> 0,
   (j + 8) := exec thalt <!> 1]

lemma tm5:
  assumes ttt = thalt * (36 + 8 * nlength thalt) + 15
  shows transforms tm5 tps0 ttt (tpsL thalt)
  unfolding tm5-def by (tform tps: jk time: assms)

lemma tm5' [transforms-intros]:
  assumes ttt = thalt * (36 + 8 * nlength thalt) + 15
  shows transforms tm5 tps0 ttt tps5
  using assms tm5 tps5-def tpsL-def by simp

definition tps6 ≡ tps0
  [(j + 1) := (⌊exec thalt <#> 0⌋N, 1),
   (j + 2) := nltape (map (λt. exec t <#> 0) [0..<Suc thalt]),
   (j + 4) := (⌊exec thalt <#> 1⌋N, 1),
   (j + 5) := nltape (map (λt. exec t <#> 1) [0..<Suc thalt]),
   (j + 7) := exec thalt <!> 0,
   (j + 8) := exec thalt <!> 1]

lemma tm6 [transforms-intros]:
  assumes ttt = thalt * (36 + 8 * nlength thalt) + 16
  shows transforms tm6 tps0 ttt tps6
  unfolding tm6-def
proof (tform tps: jk time: assms)
  show tps6 = tps5[j + 6 := tps5 ! (j + 6) |:=| 1]
    (is ?l = ?r)
  proof (rule nth-equalityI)
    show len: length ?l = length ?r
      using tps5-def tps6-def by simp
    show ?l ! i = ?r ! i if i < length ?l for i
    proof -
      have i-less: i < length ?r
        using that len by simp
      consider
          i = j | i = j + 1 | i = j + 2 | i = j + 3 | i = j + 4 | i = j + 5 | i = j + 6 | i = j + 7 | i = j + 8
        | i < j | i > j + 8
        by linarith
      then show ?thesis
        using i-less tps5-def tps6-def onesie-write tps0
        by (cases) (simp-all only: length-list-update nth-list-update-eq nth-list-update-neq)
    qed
  qed
qed
```

```
definition tps7 ≡ tps0
  [(j + 1) := (⌊exec thalt <#> 0⌋_N, 1),
   (j + 2) := (⌊map (λt. exec t <#> 0) [0..<Suc thalt]⌋_NL, 1),
   (j + 4) := (⌊exec thalt <#> 1⌋_N, 1),
   (j + 5) := nltape (map (λt. exec t <#> 1) [0..<Suc thalt]),
   (j + 7) := exec thalt <!> 0,
   (j + 8) := exec thalt <!> 1]

lemma tm7 [transforms-intros]:
  assumes ttt = thalt ∗ (36 + 8 ∗ nlength thalt) + 19 + nllength (map (λt. exec t <#> 0) [0..<Suc thalt])
  shows transforms tm7 tps0 ttt tps7
  unfolding tm7-def
proof (tform tps: tps7-def tps6-def jk assms)
  show clean-tape (tps6 ! (j + 2))
    using jk tps6-def clean-tape-nlcontents by simp
  have tps6 ! (j + 2) = nltape (map (λt. exec t <#> 0) [0..<Suc thalt])
    using jk tps6-def by simp
  then have tps6 ! (j + 2) |#=| 1 = (⌊map (λt. exec t <#> 0) [0..<Suc thalt]⌋_NL, 1) (is - = ?tp)
    by simp
  moreover have tps7 = tps6[j + 2 := ?tp]
    unfolding tps7-def tps6-def by (simp add: list-update-swap)
  ultimately show tps7 = tps6[j + 2 := tps6 ! (j + 2) |#=| 1]
    by simp
qed

definition tps8 ≡ tps0
  [(j + 1) := (⌊exec thalt <#> 0⌋_N, 1),
   (j + 2) := (⌊map (λt. exec t <#> 0) [0..<Suc thalt]⌋_NL, 1),
   (j + 4) := (⌊exec thalt <#> 1⌋_N, 1),
   (j + 5) := (⌊map (λt. exec t <#> 1) [0..<Suc thalt]⌋_NL, 1),
   (j + 7) := exec thalt <!> 0,
   (j + 8) := exec thalt <!> 1]

lemma tm8:
  assumes ttt = thalt ∗ (36 + 8 ∗ nlength thalt) + 22 + nllength (map (λt. exec t <#> 0) [0..<Suc thalt]) +
    nllength (map (λt. (exec t) <#> 1) [0..<Suc thalt])
  shows transforms tm8 tps0 ttt tps8
  unfolding tm8-def
proof (tform tps: tps8-def tps7-def jk assms)
  show clean-tape (tps7 ! (j + 5))
    using jk tps7-def clean-tape-nlcontents by simp
  have tps7 ! (j + 5) = nltape (map (λt. exec t <#> 1) [0..<Suc thalt])
    using jk tps7-def by simp
  then have tps7 ! (j + 5) |#=| 1 = (⌊map (λt. exec t <#> 1) [0..<Suc thalt]⌋_NL, 1) (is - = ?tp)
    by simp
  moreover have tps8 = tps7[j + 5 := ?tp]
    unfolding tps8-def tps7-def by (simp add: list-update-swap)
  ultimately show tps8 = tps7[j + 5 := tps7 ! (j + 5) |#=| 1]
    by simp
qed

lemma tm8':
  assumes ttt = 27 ∗ Suc thalt ∗ (9 + 2 ∗ nlength thalt)
  shows transforms tm8 tps0 ttt tps8
proof −
  have 0: nllength (map (λt. exec t <#> 0) [0..<Suc thalt]) ≤ Suc (nlength thalt) ∗ Suc thalt
    using nllength-le-len-mult-max[of map (λt. exec t <#> 0) [0..<Suc thalt] thalt] max-head-pos-0 by simp
  have 1: nllength (map (λt. exec t <#> 1) [0..<Suc thalt]) ≤ Suc (nlength thalt) ∗ Suc thalt
    using nllength-le-len-mult-max[of map (λt. exec t <#> 1) [0..<Suc thalt] thalt] max-head-pos-1 by simp

  let ?ttt = thalt ∗ (36 + 8 ∗ nlength thalt) + 22 + nllength (map (λt. exec t <#> 0) [0..<Suc thalt]) +
    nllength (map (λt. (exec t) <#> 1) [0..<Suc thalt])
  have ?ttt ≤ thalt ∗ (36 + 8 ∗ nlength thalt) + 22 + Suc (nlength thalt) ∗ Suc thalt + Suc (nlength thalt) ∗
```

```
Suc thalt
    using 0 1 by linarith
  also have ... = thalt ∗ (36 + 8 ∗ nlength thalt) + 22 + 2 ∗ Suc (nlength thalt) ∗ Suc thalt
    by simp
  also have ... ≤ Suc thalt ∗ (36 + 8 ∗ nlength thalt) + 22 + 2 ∗ Suc (nlength thalt) ∗ Suc thalt
    by simp
  also have ... ≤ Suc thalt ∗ (4 ∗ (9 + 2 ∗ nlength thalt)) + 22 + 2 ∗ Suc (nlength thalt) ∗ Suc thalt
    by simp
  also have ... = 4 ∗ Suc thalt ∗ (9 + 2 ∗ nlength thalt) + 22 + 2 ∗ Suc (nlength thalt) ∗ Suc thalt
    by linarith
  also have ... = 4 ∗ Suc thalt ∗ (9 + 2 ∗ nlength thalt) + 22 + Suc thalt ∗ (2 + 2 ∗ nlength thalt)
    by simp
  also have ... ≤ 4 ∗ Suc thalt ∗ (9 + 2 ∗ nlength thalt) + 22 + Suc thalt ∗ (9 + 2 ∗ nlength thalt)
  proof −
    have 2 + 2 ∗ nlength thalt ≤ 9 + 2 ∗ nlength thalt
      by simp
    then show ?thesis
      using Suc-mult-le-cancel1 add-le-cancel-left by blast
  qed
  also have ... = 5 ∗ Suc thalt ∗ (9 + 2 ∗ nlength thalt) + 22
    by linarith
  also have ... ≤ 5 ∗ Suc thalt ∗ (9 + 2 ∗ nlength thalt) + 22 ∗ Suc thalt ∗ (9 + 2 ∗ nlength thalt)
  proof −
    have 1 ≤ Suc thalt ∗ (9 + 2 ∗ nlength thalt)
      by simp
    then show ?thesis
      by linarith
  qed
  also have ... = 27 ∗ Suc thalt ∗ (9 + 2 ∗ nlength thalt)
    by linarith
  finally have ?ttt ≤ ttt
    using assms by simp
  then show ?thesis
    using tm8 transforms-monotone by simp
qed

end

end

lemma transforms-tm-list-headposI [transforms-intros]:
  fixes G :: nat and j :: tapeidx and M :: machine
  fixes tps tps' :: tape list
  fixes thalt k ttt :: nat and xs :: symbol list
  assumes turing-machine 2 G M
  assumes length tps = k and k ≥ j + 9 and j > 0
  assumes
    ∀t<thalt. fst (execute M (start-config 2 xs) t) < length M
    fst (execute M (start-config 2 xs) thalt) = length M
  assumes symbols-lt G xs
  assumes tps ! j = ⌈▷⌉
    tps ! (j + 1) = (⌊0⌋_N, 0)
    tps ! (j + 2) = (⌊[]⌋_NL, 0)
    tps ! (j + 3) = ⌈▷⌉
    tps ! (j + 4) = (⌊0⌋_N, 0)
    tps ! (j + 5) = (⌊[]⌋_NL, 0)
    tps ! (j + 6) = ⌈▷⌉
    tps ! (j + 7) = (⌊xs⌋, 0)
    tps ! (j + 8) = (⌊[]⌋, 0)
  assumes ttt = 27 ∗ Suc thalt ∗ (9 + 2 ∗ nlength thalt)
  assumes tps' = tps
    [(j + 1) := (⌊(execute M (start-config 2 xs) thalt) <#> 0⌋_N, 1),
     (j + 2) := (⌊map (λt. (execute M (start-config 2 xs) t) <#> 0) [0..<Suc thalt]⌋_NL, 1),
```

  (j + 4) := (⌊(execute M (start-config 2 xs) thalt) <#> 1⌋$_N$, 1),
  (j + 5) := (⌊map (λt. (execute M (start-config 2 xs) t) <#> 1) [0..<Suc thalt]⌋$_{NL}$, 1),
  (j + 7) := (execute M (start-config 2 xs) thalt) <!> 0,
  (j + 8) := (execute M (start-config 2 xs) thalt) <!> 1]
 shows *transforms (tm-list-headpos G M j) tps ttt tps'*
proof −
 interpret *loc: turing-machine-list-headpos* **.**
 show *?thesis*
  using *assms loc.tm8' loc.tm8-eq-tm-list-headpos loc.tps8-def* **by** *metis*
qed

7.4 A Turing machine for Ψ formulas

CNF formulas from the Ψ family of formulas feature prominently in Φ. In this section we first present a Turing machine for generating arbitrary members of this family and later a specialized one for the Ψ formulas that we need for Φ.

7.4.1 The general case

The next Turing machine generates a representation of the CNF formula $\Psi(vs, k)$. It expects vs as a list of numbers on tape j and the number k on tape $j + 1$. A list of lists of numbers representing $\Psi(vs, k)$ is output to tape $j + 2$.

The TM iterates over the elements of vs. In each iteration it generates a singleton clause containing the current element of vs either as positive or negative literal, depending on whether k is greater than zero or equal to zero. Then it decrements the number k. Thus the first k variable indices of vs will be turned into k positive literals, the rest into negative ones (provided $|vs| \geq k$).

```
definition tm-Psi :: tapeidx ⇒ machine where
  tm-Psi j ≡
    WHILE [] ; λrs. rs ! j ≠ □ DO
      tm-nextract | j (j + 3) ;;
      tm-times2 (j + 3) ;;
      IF λrs. rs ! (j + 1) ≠ □ THEN
        tm-incr (j + 3)
      ELSE
        []
      ENDIF ;;
      tm-to-list (j + 3) ;;
      tm-appendl (j + 2) (j + 3) ;;
      tm-decr (j + 1) ;;
      tm-erase-cr (j + 3)
    DONE ;;
    tm-cr (j + 2) ;;
    tm-erase-cr j
```

```
lemma tm-Psi-tm:
  assumes 0 < j and j + 3 < k and G ≥ 6
  shows turing-machine k G (tm-Psi j)
  unfolding tm-Psi-def
  using Nil-tm tm-nextract-tm tm-times2-tm tm-incr-tm tm-to-list-tm tm-appendl-tm tm-decr-tm
    tm-erase-cr-tm tm-cr-tm turing-machine-loop-turing-machine turing-machine-branch-turing-machine
    assms
  by simp
```

Two lemmas to help with the running time bound of *tm-Psi*:

```
lemma sum-list-mono-nth:
  fixes xs :: 'a list and f g :: 'a ⇒ nat
  assumes ∀ i<length xs. f (xs ! i) ≤ g (xs ! i)
  shows sum-list (map f xs) ≤ sum-list (map g xs)
  using assms by (metis in-set-conv-nth sum-list-mono)
```

lemma *sum-list-plus-const*:
  fixes *xs* :: *'a list* **and** *f* :: *'a* ⇒ *nat* **and** *c* :: *nat*
  shows *sum-list* (*map* (λ*x*. *c* + *f x*) *xs*) = *c* ∗ *length xs* + *sum-list* (*map f xs*)
  by (*induction xs*) *simp-all*

locale *turing-machine-Psi* =
  fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-nextract* | *j* (*j* + *3*)
definition *tm2* ≡ *tm1* ;; *tm-times2* (*j* + *3*)
definition *tm23* ≡ *IF* λ*rs*. *rs* ! (*j* + *1*) ≠ □ *THEN tm-incr* (*j* + *3*) *ELSE* [] *ENDIF*
definition *tm3* ≡ *tm2* ;; *tm23*
definition *tm4* ≡ *tm3* ;; *tm-to-list* (*j* + *3*)
definition *tm5* ≡ *tm4* ;; *tm-appendl* (*j* + *2*) (*j* + *3*)
definition *tm6* ≡ *tm5* ;; *tm-decr* (*j* + *1*)
definition *tm7* ≡ *tm6* ;; *tm-erase-cr* (*j* + *3*)
definition *tmL* ≡ *WHILE* [] ; λ*rs*. *rs* ! *j* ≠ □ *DO tm7 DONE*
definition *tm8* ≡ *tmL* ;; *tm-cr* (*j* + *2*)
definition *tm9* ≡ *tm8* ;; *tm-erase-cr j*

lemma *tm9-eq-tm-Psi*: *tm9* = *tm-Psi j*
  unfolding *tm9-def tm8-def tmL-def tm7-def tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def tm-Psi-def tm23-def*
  by *simp*

context
  fixes *tps0* :: *tape list* **and** *k kk* :: *nat* **and** *ns* :: *nat list*
  assumes *jk*: *length tps0* = *k 0* < *j j* + *3* < *k*
    and *kk*: *kk* ≤ *length ns*
  assumes *tps0*:
    tps0 ! *j* = ($\lfloor ns \rfloor_{NL}$, *1*)
    tps0 ! (*j* + *1*) = ($\lfloor kk \rfloor_N$, *1*)
    tps0 ! (*j* + *2*) = ($\lfloor [] \rfloor_{NLL}$, *1*)
    tps0 ! (*j* + *3*) = (⌊[]⌋, *1*)
begin

definition *tpsL* :: *nat* ⇒ *tape list* **where**
 tpsL t ≡ *tps0*
   [*j* := *nltape' ns t*,
    j + *1* := ($\lfloor kk - t \rfloor_N$, *1*),
    j + *2* := *nlltape* (*map* (λ*t*. [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)]) [*0*..<*t*])]

lemma *tpsL-eq-tps0*: *tpsL 0* = *tps0*
proof −
  have *tpsL 0* ! (*j* + *2*) = *tps0* ! (*j* + *2*)
    using *tpsL-def tps0 jk* **by** *simp*
  moreover have *tpsL 0* ! (*j* + *1*) = *tps0* ! (*j* + *1*)
    using *tpsL-def tps0 jk* **by** *simp*
  moreover have *tpsL 0* ! (*j* + *3*) = *tps0* ! (*j* + *3*)
    using *tpsL-def tps0 jk* **by** *simp*
  ultimately show *?thesis*
    using *tpsL-def tps0 jk*
    by (*metis* (*no-types*, *lifting*) *One-nat-def diff-zero list-update-id list-update-overwrite nllength-Nil take0*)
qed

definition *tpsL1* :: *nat* ⇒ *tape list* **where**
 tpsL1 t ≡ *tps0*
   [*j* := *nltape' ns* (*Suc t*),
    j + *1* := ($\lfloor kk - t \rfloor_N$, *1*),
    j + *2* := *nlltape* (*map* (λ*t*. [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)]) [*0*..<*t*]),
    j + *3* := ($\lfloor ns ! t \rfloor_N$, *1*)]

```
lemma tm1 [transforms-intros]:
  assumes t < length ns
    and ttt = 12 + 2 * nlength (ns ! t)
  shows transforms tm1 (tpsL t) ttt (tpsL1 t)
  unfolding tm1-def
proof (tform tps: assms(1) tpsL-def jk)
  show tpsL t ! (j + 3) = (⌊0⌋_N, 1)
    using tpsL-def jk tps0 canrepr-0 by simp
  show ttt = 12 + 2 * nlength 0 + 2 * nlength (ns ! t)
    using assms(2) by simp
  show tpsL1 t = (tpsL t)
      [j := nltape' ns (Suc t),
       j + 3 := (⌊ns ! t⌋_N, 1)]
    by (simp add: jk tps0 tpsL1-def tpsL-def list-update-swap numeral-3-eq-3)
qed

definition tpsL2 :: nat ⇒ tape list where
 tpsL2 t ≡ tps0
    [j := nltape' ns (Suc t),
     j + 1 := (⌊kk − t⌋_N, 1),
     j + 2 := nlltape (map (λt. [2 * ns ! t + (if t < kk then 1 else 0)]) [0..<t]),
     j + 3 := (⌊2 * ns ! t⌋_N, 1)]

lemma tm2 [transforms-intros]:
  assumes t < length ns
    and ttt = 17 + 4 * nlength (ns ! t)
  shows transforms tm2 (tpsL t) ttt (tpsL2 t)
  unfolding tm2-def by (tform tps: assms(1) tpsL1-def tpsL2-def jk time: assms(2))

definition tpsL3 :: nat ⇒ tape list where
 tpsL3 t ≡ tps0
    [j := nltape' ns (Suc t),
     j + 1 := (⌊kk − t⌋_N, 1),
     j + 2 := nlltape (map (λt. [2 * ns ! t + (if t < kk then 1 else 0)]) [0..<t]),
     j + 3 := (⌊2 * ns ! t + (if t < kk then 1 else 0)⌋_N, 1)]

lemma tm23 [transforms-intros]:
  assumes t < length ns
    and ttt = 7 + 2 * nlength (2 * ns ! t)
  shows transforms tm23 (tpsL2 t) ttt (tpsL3 t)
  unfolding tm23-def
proof (tform tps: assms(1) tpsL2-def jk time: assms(2))
  show read (tpsL2 t) ! (j + 1) ≠ □ ⟹
      tpsL3 t = (tpsL2 t)[j + 3 := (⌊Suc (2 * ns ! t)⌋_N, 1)]
    using tpsL2-def tpsL3-def jk read-ncontents-eq-0 by simp
  show 5 + 2 * nlength (2 * ns ! t) + 2 ≤ ttt
    using assms by simp
  show ¬ read (tpsL2 t) ! (j + 1) ≠ □ ⟹ tpsL3 t = tpsL2 t
    using tpsL2-def tpsL3-def jk read-ncontents-eq-0 by simp
qed

lemma tm3:
  assumes t < length ns
    and ttt = 24 + 4 * nlength (ns ! t) + 2 * nlength (2 * ns ! t)
  shows transforms tm3 (tpsL t) ttt (tpsL3 t)
  unfolding tm3-def by (tform tps: assms jk)

lemma tm3' [transforms-intros]:
  assumes t < length ns and ttt = 26 + 6 * nlength (ns ! t)
  shows transforms tm3 (tpsL t) ttt (tpsL3 t)
proof −
  let ?ttt = 24 + 4 * nlength (ns ! t) + 2 * nlength (2 * ns ! t)
```

have *nlength* $(2{*}x) \leq$ *Suc* (*nlength* x) **for** x
by (*metis eq-refl gr0I less-imp-le-nat nat-0-less-mult-iff nlength-0 nlength-even-le*)
then have *?ttt* $\leq 24 + 4 *$ *nlength* (*ns* ! *t*) $+ 2 *$ *Suc* (*nlength* (*ns* ! *t*))
by (*meson add-mono-thms-linordered-semiring*(*2*) *mult-le-mono2*)
then have *?ttt* $\leq 26 + 6 *$ *nlength* (*ns* ! *t*)
by *simp*
then show *?thesis*
using *tm3 assms transforms-monotone* **by** *blast*
qed

definition *tpsL4* :: *nat* $\Rightarrow$ *tape list* **where**
tpsL4 t $\equiv$ *tps0*
[j := *nltape′ ns* (*Suc t*),
$j + 1$:= ($\lfloor kk - t\rfloor_N$, *1*),
$j + 2$:= *nlltape* (*map* (λt. [$2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)]) [*0*..<*t*]),
$j + 3$:= ($\lfloor[2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)]$\rfloor_{NL}$, *1*)]

lemma *tm4*:
assumes $t <$ *length ns*
and *ttt* $= 31 + 6 *$ *nlength* (*ns* ! *t*) $+ 2 *$ *nlength* ($2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*))
shows *transforms tm4* (*tpsL t*) *ttt* (*tpsL4 t*)
unfolding *tm4-def*
proof (*tform tps*: *assms tpsL3-def jk tps0*)
show *tpsL4 t* = (*tpsL3 t*)
[$j + 3$:= ($\lfloor[2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)]$\rfloor_{NL}$, *1*)]
using *tpsL3-def tpsL4-def jk tps0* **by** *simp*
qed

lemma *tm4′* [*transforms-intros*]:
assumes $t <$ *length ns* **and** *ttt* $= 33 + 8 *$ *nlength* (*ns* ! *t*)
shows *transforms tm4* (*tpsL t*) *ttt* (*tpsL4 t*)
proof −
have *nlength* ($2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)) $\leq$ *Suc* (*nlength* (*ns* ! *t*))
using *nlength-0-simp nlength-even-le nlength-le-n nlength-times2plus1*
by (*smt* (*verit*) *add.right-neutral le-Suc-eq mult-0-right neq0-conv*)
then have $31 + 6 *$ *nlength* (*ns* ! *t*) +
$2 *$ *nlength* ($2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)) $\leq$ *ttt*
using *assms*(*2*) **by** *simp*
then show *?thesis*
using *tm4 transforms-monotone assms*(*1*) **by** *blast*
qed

definition *tpsL5* :: *nat* $\Rightarrow$ *tape list* **where**
tpsL5 t $\equiv$ *tps0*
[j := *nltape′ ns* (*Suc t*),
$j + 1$:= ($\lfloor kk - t\rfloor_N$, *1*),
$j + 2$:= *nlltape* (*map* (λt. [$2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)]) [*0*..<*Suc t*]),
$j + 3$:= ($\lfloor[2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)]$\rfloor_{NL}$, *1*)]

lemma *tm5* [*transforms-intros*]:
assumes $t <$ *length ns*
and *ttt* $= 39 + 8 *$ *nlength* (*ns* ! *t*) $+ 2 *$ *nllength* [$2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)]
shows *transforms tm5* (*tpsL t*) *ttt* (*tpsL5 t*)
unfolding *tm5-def*
proof (*tform tps*: *assms*(*1*) *tpsL4-def jk*)
let *?tps* = (*tpsL4 t*)
[$j + 2$:= *nlltape*
(*map* (λt. [$2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)]) [*0*..<*t*] @ [[$2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)]])]
show *tpsL5 t* = *?tps*
unfolding *tpsL5-def tpsL4-def*
by (*simp only*: *list-update-overwrite list-update-swap-less*[*of j+2*]) *simp*
show *ttt* $= 33 + 8 *$ *nlength* (*ns* ! *t*) +
(7 + *nlllength* (*map* (λt. [$2 *$ *ns* ! t + (*if* $t < kk$ *then 1 else 0*)]) [*0*..<*t*]) −

Suc (*nlllength* (*map* (*λt*. [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)]) [*0*..<*t*])) +
2 ∗ *nllength* [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)])
using *assms* **by** *simp*
qed

definition *tpsL6* :: *nat* ⇒ *tape list* **where**
tpsL6 t ≡ *tps0*
[*j* := *nltape′ ns* (*Suc t*),
j + *1* := (⌊*kk* − *t* − *1*⌋$_N$, *1*),
j + *2* := *nlltape* (*map* (*λt*. [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)]) [*0*..<*Suc t*]),
j + *3* := (⌊[*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)]⌋$_{NL}$, *1*)]

lemma *tm6*:
assumes *t* < *length ns*
and *ttt* = *39* + *8* ∗ *nlength* (*ns* ! *t*) +
2 ∗ *nllength* [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)] +
(*8* + *2* ∗ *nlength* (*kk* − *t*))
shows *transforms tm6* (*tpsL t*) *ttt* (*tpsL6 t*)
unfolding *tm6-def* **by** (*tform tps*: *assms*(*1*) *tpsL6-def tpsL5-def jk time*: *assms*(*2*))

lemma *nllength-elem*: *nllength* [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)] ≤ *2* + *nlength* (*ns* ! *t*)
proof −
have *2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*) ≤ *2* ∗ *ns* ! *t* + *1*
by *simp*
then have *nlength* (*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)) ≤ *nlength* (*2* ∗ *ns* ! *t* + *1*)
using *nlength-mono* **by** *simp*
then have *nlength* (*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)) ≤ *Suc* (*nlength* (*ns* ! *t*))
using *nlength-times2plus1* **by** *fastforce*
then show *?thesis*
using *nllength* **by** *simp*
qed

lemma *tm6′* [*transforms-intros*]:
assumes *t* < *length ns*
and *ttt* = *43* + *10* ∗ *nlength* (*ns* ! *t*) + (*8* + *2* ∗ *nlength* (*kk* − *t*))
shows *transforms tm6* (*tpsL t*) *ttt* (*tpsL6 t*)
proof −
let *?ttt* = *39* + *8* ∗ *nlength* (*ns* ! *t*) +
2 ∗ *nllength* [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)] +
(*8* + *2* ∗ *nlength* (*kk* − *t*))
have *?ttt* ≤ *39* + *8* ∗ *nlength* (*ns* ! *t*) +
2 ∗ (*2* + *nlength* (*ns* ! *t*)) + (*8* + *2* ∗ *nlength* (*kk* − *t*))
using *nllength-elem*
by (*meson add-mono-thms-linordered-semiring*(*2*) *add-mono-thms-linordered-semiring*(*3*) *nat-mult-le-cancel-disj*)
also have ... ≤ *43* + *10* ∗ *nlength* (*ns* ! *t*) + (*8* + *2* ∗ *nlength* (*kk* − *t*))
by *simp*
finally have *?ttt* ≤ *ttt*
using *assms*(*2*) **by** *simp*
then show *?thesis*
using *assms*(*1*) *tm6 transforms-monotone* **by** *blast*
qed

definition *tpsL7* :: *nat* ⇒ *tape list* **where**
tpsL7 t ≡ *tps0*
[*j* := *nltape′ ns* (*Suc t*),
j + *1* := (⌊*kk* − *Suc t*⌋$_N$, *1*),
j + *2* := *nlltape* (*map* (*λt*. [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)]) [*0*..<*Suc t*]),
j + *3* := (⌊[]⌋, *1*)]

lemma *tm7*:
assumes *t* < *length ns*
and *ttt* = *51* + (*10* ∗ *nlength* (*ns* ! *t*) + *2* ∗ *nlength* (*kk* − *t*)) +
(*7* + *2* ∗ *length* (*numlist* [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)]))

```
  shows transforms tm7 (tpsL t) ttt (tpsL7 t)
  unfolding tm7-def
proof (tform tps: assms(1) tpsL6-def tpsL7-def jk time: tpsL6-def jk assms(2))
  let ?ns = [2 * ns ! t + (if t < kk then 1 else 0)]
  show tpsL6 t ::: (j + 3) = ⌊numlist ?ns⌋
    using tpsL6-def nlcontents-def jk by simp
  show proper-symbols (numlist ?ns)
    using proper-symbols-numlist by simp
qed

lemma tm7':
  assumes t < length ns and ttt = 62 + 14 * nllength ns
  shows transforms tm7 (tpsL t) ttt (tpsL7 t)
proof -
  let ?ttt = 51 + (10 * nlength (ns ! t) + 2 * nlength (kk - t)) +
      (7 + 2 * length (numlist [2 * ns ! t + (if t < kk then 1 else 0)]))
  have ?ttt = 58 + (10 * nlength (ns ! t) + 2 * nlength (kk - t)) +
      2 * length (numlist [2 * ns ! t + (if t < kk then 1 else 0)])
    by simp
  also have ... ≤ 58 + (10 * nlength (ns ! t) + 2 * nlength (kk - t)) + 2 * (2 + nlength (ns ! t))
    using nllength-elem nllength-def mult-le-mono2 nat-add-left-cancel-le by metis
  also have ... = 62 + 12 * nlength (ns ! t) + 2 * nlength (kk - t)
    by simp
  also have ... ≤ 62 + 12 * nlength (ns ! t) + 2 * nlength (length ns)
    using assms(1) kk nlength-mono by simp
  also have ... ≤ 62 + 12 * nllength ns + 2 * nlength (length ns)
    using assms(1) member-le-nllength by simp
  also have ... ≤ 62 + 12 * nllength ns + 2 * nllength ns
    using length-le-nllength nlength-le-n by (meson add-left-mono dual-order.trans mult-le-mono2)
  also have ... = 62 + 14 * nllength ns
    by simp
  finally have ?ttt ≤ 62 + 14 * nllength ns .
  then show ?thesis
    using assms tm7 transforms-monotone by blast
qed

lemma tpsL7-eq-tpsL: tpsL7 t = tpsL (Suc t)
  unfolding tpsL7-def tpsL-def
  using jk tps0
  by (smt (verit) Suc-eq-plus1 add-2-eq-Suc' add-cancel-left-right add-left-cancel list-update-id list-update-swap
   num.simps(8) numeral-eq-iff numeral-eq-one-iff semiring-norm(86) zero-neq-numeral)

lemma tm7'' [transforms-intros]:
  assumes t < length ns and ttt = 62 + 14 * nllength ns
  shows transforms tm7 (tpsL t) ttt (tpsL (Suc t))
  using assms tpsL7-eq-tpsL tm7' by simp

lemma tmL [transforms-intros]:
  assumes ttt = length ns * (62 + 14 * nllength ns + 2) + 1
  shows transforms tmL (tpsL 0) ttt (tpsL (length ns))
  unfolding tmL-def
proof (tform)
  let ?t = 62 + 14 * nllength ns
  show read (tpsL t) ! j ≠ □ if t < length ns for t
  proof -
    have tpsL t ! j = nltape' ns t
      using tpsL-def jk by simp
    then show ?thesis
      using nltape'-tape-read that tapes-at-read' tpsL-def jk
      by (metis (no-types, lifting) add-lessD1 leD length-list-update)
  qed
  have tpsL t ! j = nltape' ns t for t
    using tpsL-def jk nlcontents-def by simp
```

then show ¬ *read* (*tpsL* (*length ns*)) ! *j* ≠ □
  using *nltape′-tape-read tpsL-def jk tapes-at-read′*
  by (*metis* (*no-types*, *lifting*) *add-lessD1 length-list-update order-refl*)
show *length ns* ∗ (*62* + *14* ∗ *nllength ns* + *2*) + *1* ≤ *ttt*
  using *assms* **by** *simp*
qed

definition *tps8* :: *tape list* **where**
tps8 ≡ *tps0*
[*j* := *nltape′ ns* (*length ns*),
j + *1* := (⌊*0*⌋$_N$, *1*),
j + *2* := *nlltape′* (*map* (λ*t*. [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)]) [*0*..<*length ns*]) *0*]

lemma *tm8*:
assumes *ttt* = *Suc* (*length ns* ∗ (*64* + *14* ∗ *nllength ns*)) +
Suc (*Suc* (*Suc* (*nlllength* (*map* (λ*t*. [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)]) [*0*..<*length ns*]))))
shows *transforms tm8* (*tpsL 0*) *ttt tps8*
unfolding *tm8-def*
proof (*tform tps*: *tpsL-def tps8-def jk time*: *assms tpsL-def jk*)
show *clean-tape* (*tpsL* (*length ns*) ! (*j* + *2*))
  using *tpsL-def jk clean-tape-nllcontents* **by** *simp*
show *tps8* = (*tpsL* (*length ns*))[*j* + *2* := *tpsL* (*length ns*) ! (*j* + *2*) |#=| *1*]
  unfolding *tps8-def tpsL-def* **using** *jk kk* **by** *simp*
qed

lemma *tm8′* [*transforms-intros*]:
assumes *ttt* = *4* + *81* ∗ *nllength ns* ^ *2*
shows *transforms tm8 tps0 ttt tps8*
proof −
let *?nss* = *map* (λ*t*. [*2* ∗ *ns* ! *t* + (*if t* < *kk then 1 else 0*)]) [*0*..<*length ns*]
let *?ttt* = *Suc* (*length ns* ∗ (*64* + *14* ∗ *nllength ns*)) + *Suc* (*Suc* (*Suc* (*nlllength ?nss*)))

have *nlllength ?nss* = (∑*ns*←*?nss*. *Suc* (*nllength ns*))
  using *nlllength* **by** *simp*
also have ... = (∑*i*←[*0*..<*length ns*]. *Suc* (*nllength* [*2* ∗ *ns* ! *i* + (*if i* < *kk then 1 else 0*)]))
proof −
  have *map* (λ*ns*. *Suc* (*nllength ns*)) *?nss* = *map* (λ*i*. *Suc* (*nllength* (*?nss* ! *i*))) [*0*..<*length ns*]
    by *simp*
  then have *map* (λ*ns*. *Suc* (*nllength ns*)) *?nss* = *map* (λ*i*. *Suc* (*nllength* [*2* ∗ *ns* ! *i* + (*if i* < *kk then 1 else 0*)])) [*0*..<*length ns*]
    by *simp*
  then show *?thesis*
    by *metis*
qed
also have ... = (∑*i*←[*0*..<*length ns*]. *Suc* (*Suc* (*nlength* (*2* ∗ *ns* ! *i* + (*if i* < *kk then 1 else 0*)))))
  using *nllength* **by** *simp*
also have ... ≤ (∑*i*←[*0*..<*length ns*]. *Suc* (*Suc* (*nlength* (*2* ∗ *ns* ! *i* + *1*))))
  using *sum-list-mono-nth*[*of* [*0*..<*length ns*]] *nlength-mono* **by** *simp*
also have ... ≤ (∑*i*←[*0*..<*length ns*]. *Suc* (*Suc* (*Suc* (*nlength* (*ns* ! *i*)))))
  using *sum-list-mono-nth*[*of* [*0*..<*length ns*]] *nlength-times2plus1* **by** *simp*
also have ... = (∑*i*←[*0*..<*length ns*]. *2* + *Suc* (*nlength* (*ns* ! *i*)))
  by *simp*
also have ... = *2* ∗ *length ns* + (∑*i*←[*0*..<*length ns*]. *Suc* (*nlength* (*ns* ! *i*)))
  using *sum-list-plus-const*[*of 2* - [*0*..<*length ns*]] **by** *simp*
also have ... = *2* ∗ *length ns* + *nllength ns*
proof −
  have *map* (λ*i*. *Suc* (*nlength* (*ns* ! *i*))) [*0*..<*length ns*] = *map* (λ*n*. *Suc* (*nlength n*)) *ns*
    by (*rule nth-equalityI*) *simp-all*
  then show *?thesis*
    using *nllength* **by** *simp*
qed
finally have *nlllength ?nss* ≤ *2* ∗ *length ns* + *nllength ns* **.**
then have *?ttt* ≤ *Suc* (*length ns* ∗ (*64* + *14* ∗ *nllength ns*)) + *Suc* (*Suc* (*Suc* (*2* ∗ *length ns* + *nllength ns*)))

by *simp*
also have ... = $4 + length\ ns * (64 + 14 * nllength\ ns) + 2 * length\ ns + nllength\ ns$
by *simp*
also have ... = $4 + length\ ns * (66 + 14 * nllength\ ns) + nllength\ ns$
by *algebra*
also have ... $\leq 4 + nllength\ ns * (66 + 14 * nllength\ ns) + nllength\ ns$
using *length-le-nllength* by *simp*
also have ... = $4 + 67 * nllength\ ns + 14 * nllength\ ns\ \hat{}\ 2$
by *algebra*
also have ... $\leq 4 + 67 * nllength\ ns\ \hat{}\ 2 + 14 * nllength\ ns\ \hat{}\ 2$
using *linear-le-pow* by *simp*
also have ... = $4 + 81 * nllength\ ns\ \hat{}\ 2$
by *simp*
finally have $?ttt \leq 4 + 81 * nllength\ ns\ \hat{}\ 2$ **.**
then show *?thesis*
using *tm8 assms transforms-monotone tpsL-eq-tps0* by *simp*
qed

definition *tps9* :: *tape list* **where**
$tps9 \equiv tps0$
$[j := (\lfloor[]\rfloor, 1),$
$j + 1 := (\lfloor 0\rfloor_N, 1),$
$j + 2 := nlltape'\ (map\ (\lambda t.\ [2 * ns\ !\ t + (if\ t < kk\ then\ 1\ else\ 0)])\ [0..<length\ ns])\ 0]$

lemma *tm9*:
assumes $ttt = 11 + 81 * nllength\ ns\ \hat{}\ 2 + 3 * nllength\ ns$
shows *transforms tm9 tps0 ttt tps9*
unfolding *tm9-def*
proof (*tform tps*: *tps8-def tps9-def jk*)
show $tps8 ::: j = \lfloor numlist\ ns\rfloor$
using *tps8-def jk nlcontents-def* **by** *simp*
show *proper-symbols* (*numlist ns*)
using *proper-symbols-numlist* **by** *simp*
show $ttt = 4 + 81 * (nllength\ ns)^2 + (tps8 :\#: j + 2 * length\ (numlist\ ns) + 6)$
using *tps8-def jk assms nllength-def* **by** *simp*
qed

lemma *tm9'*:
assumes $ttt = 11 + 84 * nllength\ ns\ \hat{}\ 2$
shows *transforms tm9 tps0 ttt tps9*
proof −
have $11 + 81 * nllength\ ns\ \hat{}\ 2 + 3 * nllength\ ns \leq 11 + 84 * nllength\ ns\ \hat{}\ 2$
using *linear-le-pow* **by** *simp*
then show *?thesis*
using *tm9 assms transforms-monotone* **by** *simp*
qed

end

end

lemma *transforms-tm-PsiI* [*transforms-intros*]:
fixes *j* :: *tapeidx*
fixes *tps tps'* :: *tape list* **and** *ttt k kk* :: *nat* **and** *ns* :: *nat list*
assumes $length\ tps = k\ 0 < j\ j + 3 < k$
and $kk \leq length\ ns$
assumes
$tps\ !\ j = (\lfloor ns\rfloor_{NL}, 1)$
$tps\ !\ (j + 1) = (\lfloor kk\rfloor_N, 1)$
$tps\ !\ (j + 2) = (\lfloor[]\rfloor_{NLL}, 1)$
$tps\ !\ (j + 3) = (\lfloor[]\rfloor, 1)$
assumes $ttt = 11 + 84 * nllength\ ns\ \hat{}\ 2$
assumes $tps' = tps$

$[j := (\lfloor[]\rfloor, 1),$
$j + 1 := (\lfloor 0\rfloor_N, 1),$
$j + 2 := nlltape' (map (\lambda t. [2 * ns ! t + (if\ t < kk\ then\ 1\ else\ 0)]) [0..<length\ ns]) 0]$
shows *transforms (tm-Psi j) tps ttt tps'*
proof −
interpret *loc*: *turing-machine-Psi j* **.**
show *?thesis*
using *loc.tm9-eq-tm-Psi loc.tps9-def loc.tm9' assms* **by** *simp*
qed

7.4.2 For intervals

To construct Φ we need only Ψ formulas where the variable index list is an interval $\gamma_i = [iH, (i+1)H)$. In this section we devise a Turing machine that outputs $\Psi([start, start+len), \kappa)$ for arbitrary $start$, len, and κ.

definition *nll-Psi* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat list list* **where**
nll-Psi start len κ ≡ *map* (*λi.* [*2* ∗ (*start* + *i*) + (*if i* < *κ then 1 else 0*)]) [*0*..<*len*]

lemma *nll-Psi*: *nll-Psi start len κ* = *formula-n* (Ψ [*start*..<*start*+*len*] *κ*)
(**is** *?lhs* = *?rhs*)
proof (*rule nth-equalityI*)
show *len*: *length ?lhs* = *length ?rhs*
using *nll-Psi-def Psi-def formula-n-def* **by** *simp*
let *?psi* = Ψ [*start*..<*start*+*len*] *κ*
show *?lhs* ! *i* = *?rhs* ! *i* **if** *i* < *length ?lhs* **for** *i*
proof −
have *i* < *length ?psi*
using *that Psi-def nll-Psi-def* **by** *simp*
have *i* < *len*
using *that Psi-def nll-Psi-def* **by** *simp*
show *?thesis*
proof (*cases i* < *κ*)
case *True*
then have *?psi* ! *i* = [*Pos* (*start* + *i*)]
using *Psi-def nth-append*[*of map* (*λs.* [*Pos s*]) (*take κ* [*start*..<*start*+*len*]) - *i*] ‹*i* < *len*›
by *simp*
moreover have *?rhs* ! *i* = *clause-n* (*?psi* ! *i*)
using *formula-n-def that* ‹*i* < *length ?psi*› **by** *simp*
ultimately have *?rhs* ! *i* = [*Suc* (*2* ∗ (*start* + *i*))]
using *clause-n-def* **by** *simp*
moreover have *?lhs* ! *i* = [*2* ∗ (*start* + *i*) + *1*]
using *True nll-Psi-def that* **by** *simp*
ultimately show *?thesis*
by *simp*
next
case *False*
then have *?psi* ! *i* = [*Neg* (*start* + *i*)]
using *Psi-def nth-append*[*of map* (*λs.* [*Pos s*]) (*take κ* [*start*..<*start*+*len*]) - *i*] ‹*i* < *len*›
by *auto*
moreover have *?rhs* ! *i* = *clause-n* (*?psi* ! *i*)
using *formula-n-def that* ‹*i* < *length ?psi*› **by** *simp*
ultimately have *?rhs* ! *i* = [*2* ∗ (*start* + *i*)]
using *clause-n-def* **by** *simp*
moreover have *?lhs* ! *i* = [*2* ∗ (*start* + *i*)]
using *False nll-Psi-def that* **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed
qed

lemma *nlllength-nll-Psi-le*: *nlllength* (*nll-Psi start len κ*) ≤ *len* ∗ (*3* + *nlength* (*start* + *len*))
proof −

define *f* :: *nat* ⇒ *nat list* **where**
f = (λ*i*. [*2* ∗ (*start* + *i*) + (*if i* < κ *then 1 else 0*)])
let *?nss* = *map f* [*0*..<*len*]
have *nlllength* (*nll-Psi start len* κ) = ($\sum$ *ns*←*?nss*. *Suc* (*nllength ns*))
using *nlllength f-def nll-Psi-def* **by** *simp*
also have ... = ($\sum$ *i*←[*0*..<*len*]. (λ*ns*. *Suc* (*nllength ns*)) (*f i*))
by (*metis* (*no-types*, *lifting*) *map-eq-conv map-map o-apply*)
also have ... = ($\sum$ *i*←[*0*..<*len*]. *Suc* (*nllength* ([*2* ∗ (*start* + *i*) + (*if i* < κ *then 1 else 0*)])))
using *f-def* **by** *simp*
also have ... = ($\sum$ *i*←[*0*..<*len*]. *Suc* (*Suc* (*nlength* (*2* ∗ (*start* + *i*) + (*if i* < κ *then 1 else 0*)))))
using *nllength* **by** *simp*
also have ... ≤ ($\sum$ *i*←[*0*..<*len*]. *Suc* (*Suc* (*nlength* (*2* ∗ (*start* + *i*) + *1*))))
using *nlength-mono*
sum-list-mono[*of* [*0*..<*len*]
λ*i*. *Suc* (*Suc* (*nlength* (*2* ∗ (*start* + *i*) + (*if i* < κ *then 1 else 0*))))
λ*i*. *Suc* (*Suc* (*nlength* (*2* ∗ (*start* + *i*) + *1*)))]
by *simp*
also have ... ≤ ($\sum$ *i*←[*0*..<*len*]. *Suc* (*Suc* (*nlength* (*2* ∗ (*start* + *len*)))))
using *nlength-mono*
sum-list-mono[*of* [*0*..<*len*]
λ*i*. *Suc* (*Suc* (*nlength* (*2* ∗ (*start* + *i*) + *1*)))
λ*i*. *Suc* (*Suc* (*nlength* (*2* ∗ (*start* + *len*))))]
by *simp*
also have ... = *len* ∗ *Suc* (*Suc* (*nlength* (*2* ∗ (*start* + *len*))))
using *sum-list-const*[*of* - [*0*..<*len*]] **by** *simp*
also have ... ≤ *len* ∗ *Suc* (*Suc* (*Suc* (*nlength* (*start* + *len*))))
using *nlength-times2* **by** (*metis add-gr-0 eq-refl mult-le-cancel1 nlength-even-le*)
also have ... = *len* ∗ (*3* + *nlength* (*start* + *len*))
by (*simp add*: *Suc3-eq-add-3*)
finally show *?thesis* **.**
qed

lemma *nlllength-nll-Psi-le′*:
assumes *start1* ≤ *start2*
shows *nlllength* (*nll-Psi start1 len* κ) ≤ *len* ∗ (*3* + *nlength* (*start2* + *len*))
proof −
have *nlllength* (*nll-Psi start1 len* κ) ≤ *len* ∗ (*3* + *nlength* (*start1* + *len*))
using *nlllength-nll-Psi-le* **by** *simp*
moreover have *nlength* (*start1* + *len*) ≤ *nlength* (*start2* + *len*)
using *assms nlength-mono* **by** *simp*
ultimately show *?thesis*
by (*meson add-mono-thms-linordered-semiring*(*2*) *mult-le-mono2 order-trans*)
qed

lemma *H4-nlength*:
fixes *x y H* :: *nat*
assumes *x* ≤ *y* **and** *H* ≥ *3*
shows *H* ^ *4* ∗ (*nlength x*)^2 ≤ *H* ^ *4* ∗ (*nlength y*)^2
using *assms* **by** (*simp add*: *nlength-mono*)

The next Turing machine receives on tape j a number i, on tape $j + 1$ a number H, and on tape $j + 2$ a number κ. It outputs $\Psi([i \cdot H, (i + 1) \cdot H), \kappa)$.

definition *tm-Psigamma* :: *tapeidx* ⇒ *machine* **where**
tm-Psigamma j ≡
tm-mult j (*j* + *1*) (*j* + *3*) ;;
tm-range (*j* + *3*) (*j* + *1*) (*j* + *4*) ;;
tm-copyn (*j* + *2*) (*j* + *5*) ;;
tm-Psi (*j* + *4*) ;;
tm-erase-cr (*j* + *3*)

lemma *tm-Psigamma-tm*:
assumes *G* ≥ *6* **and** *j* + *7* < *k*
shows *turing-machine k G* (*tm-Psigamma j*)

unfolding *tm-Psigamma-def*
using *assms tm-mult-tm tm-range-tm tm-copyn-tm tm-Psi-tm tm-erase-cr-tm*
by *simp*

locale *turing-machine-Psigamma* =
fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-mult j* (*j* + *1*) (*j* + *3*)
definition *tm2* ≡ *tm1* ;; *tm-range* (*j* + *3*) (*j* + *1*) (*j* + *4*)
definition *tm3* ≡ *tm2* ;; *tm-copyn* (*j* + *2*) (*j* + *5*)
definition *tm4* ≡ *tm3* ;; *tm-Psi* (*j* + *4*)
definition *tm5* ≡ *tm4* ;; *tm-erase-cr* (*j* + *3*)

lemma *tm5-eq-tm-Psigamma*: *tm5* = *tm-Psigamma j*
using *tm5-def tm4-def tm3-def tm2-def tm1-def tm-Psigamma-def* **by** *simp*

context
fixes *tps0* :: *tape list* **and** *H k idx* κ :: *nat*
assumes *jk*: *length tps0* = *k 0* < *j j* + *7* < *k*
and *H*: *H* ≥ *3*
and κ: $\kappa \leq H$
assumes *tps0*:
tps0 ! *j* = (⌊*idx*⌋$_N$, *1*)
tps0 ! (*j* + *1*) = (⌊*H*⌋$_N$, *1*)
tps0 ! (*j* + *2*) = (⌊κ⌋$_N$, *1*)
tps0 ! (*j* + *3*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *4*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *5*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *6*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *7*) = (⌊[]⌋, *1*)
begin

definition *tps1* ≡ *tps0*
[*j* + *3* := (⌊*idx* ∗ *H*⌋$_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *4* + *26* ∗ (*nlength idx* + *nlength H*) ^ *2*
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
proof (*tform tps*: *jk tps0 tps1-def*)
show *tps0* ! (*j* + *3*) = (⌊*0*⌋$_N$, *1*)
using *tps0 canrepr-0* **by** *simp*
show *ttt* = *4* + *26* ∗ (*nlength idx* + *nlength H*) ∗ (*nlength idx* + *nlength H*)
using *assms* **by** *algebra*
qed

definition *tps2* ≡ *tps0*
[*j* + *3* := (⌊*idx* ∗ *H*⌋$_N$, *1*),
j + *4* := (⌊[*idx* ∗ *H*..<*idx* ∗ *H* + *H*]⌋$_{NL}$, *1*)]

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *4* + *26* ∗ (*nlength idx* + *nlength H*)^2 + *Suc H* ∗ (*43* + *9* ∗ *nlength* (*idx* ∗ *H* + *H*))
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def*
proof (*tform tps*: *tps0 tps1-def tps2-def jk time*: *assms*)
show *tps1* ! (*j* + *4*) = (⌊[]⌋$_{NL}$, *1*)
using *tps1-def tps0 jk nlcontents-Nil* **by** *simp*
have *j* + *4* + *1* = *j* + *5*
by *simp*
then show *tps1* ! (*j* + *4* + *1*) = (⌊*0*⌋$_N$, *1*)
using *tps1-def tps0 jk canrepr-0*
by (*metis add-left-imp-eq nth-list-update-neq numeral-eq-iff semiring-norm*(*83*) *semiring-norm*(*90*))

have $j + 4 + 2 = j + 6$
by *simp*
then show $tps1 ! (j + 4 + 2) = (\lfloor 0 \rfloor_N, 1)$
using *tps1-def tps0 jk canrepr-0*
by (*metis add-left-imp-eq nth-list-update-neq num.simps*(*8*) *numeral-eq-iff*)
qed

definition $tps3 \equiv tps0$
$[j + 3 := (\lfloor idx * H \rfloor_N, 1),$
$j + 4 := (\lfloor [idx * H..<idx * H + H] \rfloor_{NL}, 1),$
$j + 5 := (\lfloor \kappa \rfloor_N, 1)]$

lemma *tm3* [*transforms-intros*]:
assumes $ttt = 18 + 26 * (nlength\ idx + nlength\ H)^2 + Suc\ H * (43 + 9 * nlength\ (idx * H + H)) + 3 * nlength\ \kappa$
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def*
proof (*tform tps*: *tps0 tps2-def tps3-def jk*)
show $tps2 ! (j + 5) = (\lfloor 0 \rfloor_N, 1)$
using *tps2-def jk tps0 canrepr-0* **by** *simp*
show $ttt = 4 + 26 * (nlength\ idx + nlength\ H)^2 +$
$Suc\ H * (43 + 9 * nlength\ (idx * H + H)) +$
$(14 + 3 * (nlength\ \kappa + nlength\ 0))$
using *assms* **by** *simp*
qed

definition $tps4 \equiv tps0$
$[j + 3 := (\lfloor idx * H \rfloor_N, 1),$
$j + 4 := (\lfloor [] \rfloor, 1),$
$j + 5 := (\lfloor 0 \rfloor_N, 1),$
$j + 6 := nlltape'$
$(map\ (\lambda t.\ [2 * [idx * H..<idx * H + H] ! t + (if\ t < \kappa\ then\ 1\ else\ 0)])$
$[0..<length\ [idx * H..<idx * H + H]])\ 0]$

lemma *tm4*:
assumes $ttt = 29 + 26 * (nlength\ idx + nlength\ H)^2 + Suc\ H * (43 + 9 * nlength\ (idx * H + H)) +$
$3 * nlength\ \kappa + 84 * (nllength\ [idx * H..<idx * H + H])^2$
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform tps*: *tps0 tps3-def tps4-def jk* κ *time*: *assms*)
have $*$: $j + 4 + 1 = j + 5\ j + 4 + 2 = j + 6\ j + 4 + 3 = j + 7$
using *add.assoc* **by** *simp-all*
have $tps3 ! (j + 5) = (\lfloor \kappa \rfloor_N, 1)$
using *tps3-def jk* **by** *simp*
then show $tps3 ! (j + 4 + 1) = (\lfloor \kappa \rfloor_N, 1)$
using $*$ **by** *metis*
have $tps3 ! (j + 6) = (\lfloor [] \rfloor_{NLL}, 1)$
using *tps3-def jk tps0 nllcontents-Nil* **by** *simp*
then show $tps3 ! (j + 4 + 2) = (\lfloor [] \rfloor_{NLL}, 1)$
using $*$ **by** *metis*
have $tps3 ! (j + 7) = (\lfloor [] \rfloor, 1)$
using *tps3-def jk tps0* **by** *simp*
then show $tps3 ! (j + 4 + 3) = (\lfloor [] \rfloor, 1)$
using $*$ **by** *metis*
have $tps4 = tps3$
$[j + 4 := (\lfloor [] \rfloor, 1),$
$j + 5 := (\lfloor 0 \rfloor_N, 1),$
$j + 6 := nlltape'$
$(map\ (\lambda t.\ [2 * [idx * H..<idx * H + H] ! t + (if\ t < \kappa\ then\ 1\ else\ 0)])$
$[0..<length\ [idx * H..<idx * H + H]])\ 0]$
unfolding *tps4-def tps3-def*
by (*simp only*: *list-update-overwrite list-update-swap-less*)
then show $tps4 = tps3$

$[j + 4 := (\lfloor[]\rfloor, 1),$
$j + 4 + 1 := (\lfloor 0\rfloor_N, 1),$
$j + 4 + 2 := nlltape'$
$(map\ (\lambda t.\ [2 * [idx * H..<idx * H + H]\ !\ t + (if\ t < \kappa\ then\ 1\ else\ 0)])$
$[0..<length\ [idx * H..<idx * H + H]])\ 0]$
using $*$ **by** *metis*
qed

definition $tps4' \equiv tps0$
$[j + 3 := (\lfloor idx * H\rfloor_N, 1),$
$j + 4 := (\lfloor[]\rfloor, 1),$
$j + 5 := (\lfloor 0\rfloor_N, 1),$
$j + 6 := nlltape'\ (map\ (\lambda t.\ [2 * (idx * H + t) + (if\ t < \kappa\ then\ 1\ else\ 0)])\ [0..<H])\ 0]$

lemma *tps4'-eq-tps4*: $tps4' = tps4$
proof −
have $map\ (\lambda t.\ [2 * [idx * H..<idx * H + H]\ !\ t + (if\ t < \kappa\ then\ 1\ else\ 0)])\ [0..<length\ [idx * H..<idx * H + H]] =$
$map\ (\lambda t.\ [2 * (idx * H + t) + (if\ t < \kappa\ then\ 1\ else\ 0)])\ [0..<H]$
by *simp*
then show *?thesis*
using *tps4'-def tps4-def* **by** *metis*
qed

lemma *tm4'* [*transforms-intros*]:
assumes $ttt = 29 + 26 * (nlength\ idx + nlength\ H)^2 + Suc\ H * (43 + 9 * nlength\ (idx * H + H)) +$
$3 * nlength\ \kappa + 84 * (nllength\ [idx * H..<idx * H + H])^2$
shows *transforms tm4 tps0 ttt tps4'*
using *tm4 tps4'-eq-tps4 assms* **by** *simp*

definition $tps5 \equiv tps0$
$[j + 3 := (\lfloor[]\rfloor, 1),$
$j + 4 := (\lfloor[]\rfloor, 1),$
$j + 5 := (\lfloor 0\rfloor_N, 1),$
$j + 6 := nlltape'\ (map\ (\lambda t.\ [2 * (idx * H + t) + (if\ t < \kappa\ then\ 1\ else\ 0)])\ [0..<H])\ 0]$

lemma *tm5*:
assumes $ttt = 36 + 26 * (nlength\ idx + nlength\ H)^2 + Suc\ H * (43 + 9 * nlength\ (idx * H + H)) +$
$3 * nlength\ \kappa + 84 * (nllength\ [idx * H..<idx * H + H])^2 +$
$2 * nlength\ (idx * H)$
shows *transforms tm5 tps0 ttt tps5*
unfolding *tm5-def*
proof (*tform tps*: *tps0 tps4'-def tps5-def jk* κ *time*: *assms tps4'-def jk*)
show $proper\text{-}symbols\ (canrepr\ (idx * H))$
using *proper-symbols-canrepr* **by** *simp*
qed

definition $tps5' \equiv tps0$
$[j + 6 := (\lfloor nll\text{-}Psi\ (idx * H)\ H\ \kappa\rfloor_{NLL}, 1)]$

lemma *tps5'-eq-tps5*: $tps5' = tps5$
using *tps5'-def tps5-def jk tps0 nll-Psi-def nlltape'-0 canrepr-0* **by** (*metis list-update-id*)

lemma *tm5'*:
assumes $ttt = 1851 * H\hat{}4 * (nlength\ (Suc\ idx))\hat{}2$
shows *transforms tm5 tps0 ttt tps5'*
proof −
let $?ttt = 36 + 26 * (nlength\ idx + nlength\ H)^2 + Suc\ H * (43 + 9 * nlength\ (idx * H + H)) +$
$3 * nlength\ \kappa + 84 * (nllength\ [idx * H..<idx * H + H])^2 + 2 * nlength\ (idx * H)$
have $?ttt \le 36 + 26 * (nlength\ idx + nlength\ H)^2 + Suc\ H * (43 + 9 * nlength\ (idx * H + H)) +$
$3 * nlength\ \kappa + 84 * (Suc\ (nlength\ (Suc\ idx * H)) * H)^2 + 2 * nlength\ (idx * H)$
using *nllength-le-len-mult-max*[*of* $[idx * H..<idx * H + H]$ $Suc\ idx * H$] **by** *simp*
also have ... $\le 36 + 26 * (nlength\ idx + nlength\ H)^2 + Suc\ H * (43 + 9 * nlength\ (idx * H + H)) +$

3 ∗ nlength H + 84 ∗ (Suc (nlength (Suc idx ∗ H)) ∗ H)^2 *+ 2 ∗ nlength (idx ∗ H)*
using *κ nlength-mono* **by** *simp*
also have ... *= 36 + 26 ∗ (nlength idx + nlength H)*^{2} *+ Suc H ∗ (43 + 9 ∗ nlength (Suc idx ∗ H)) +*
3 ∗ nlength H + 84 ∗ (Suc (nlength (Suc idx ∗ H)) ∗ H)^2 *+ 2 ∗ nlength (idx ∗ H)*
by (*simp add*: *add.commute*)
also have ... *≤ 36 + 26 ∗ (nlength idx + nlength H)*^{2} *+ Suc H ∗ (43 + 9 ∗ (nlength (Suc idx) + nlength H))*
+
3 ∗ nlength H + 84 ∗ (Suc (nlength (Suc idx ∗ H)) ∗ H)^2 *+ 2 ∗ nlength (idx ∗ H)*
proof −
have *Suc H ∗ (43 + 9 ∗ nlength (Suc idx ∗ H)) ≤ Suc H ∗ (43 + 9 ∗ (nlength (Suc idx) + nlength H))*
using *nlength-prod* **by** (*meson add-mono-thms-linordered-semiring(2) mult-le-mono2*)
then show *?thesis*
by *simp*
qed
also have ... *≤ 36 + 26 ∗ (nlength idx + nlength H)*^{2} *+ Suc H ∗ (43 + 9 ∗ (nlength (Suc idx) + nlength H))*
+
3 ∗ nlength H + 84 ∗ (Suc (nlength (Suc idx) + nlength H) ∗ H)^2 *+ 2 ∗ nlength (idx ∗ H)*
using *nlength-prod Suc-le-mono add-le-mono le-refl mult-le-mono power2-nat-le-eq-le* **by** *presburger*
also have ... *≤ 36 + 26 ∗ (nlength idx + nlength H)*^{2} *+ Suc H ∗ (43 + 9 ∗ (nlength (Suc idx) + nlength H))*
+
3 ∗ nlength H + 84 ∗ (Suc (nlength (Suc idx) + nlength H) ∗ H)^2 *+ 2 ∗ (nlength idx + nlength H)*
using *nlength-prod Suc-le-mono add-le-mono le-refl mult-le-mono power2-nat-le-eq-le* **by** *presburger*
also have ... *≤ 36 + 26 ∗ (Suc (nlength (Suc idx) + nlength H) ∗ H)*^{2} *+ Suc H ∗ (43 + 9 ∗ (nlength (Suc idx) + nlength H)) +*
3 ∗ nlength H + 84 ∗ (Suc (nlength (Suc idx) + nlength H) ∗ H)^2 *+ 2 ∗ (nlength idx + nlength H)*
proof −
have *nlength idx + nlength H ≤ Suc (nlength (Suc idx) + nlength H)*
using *nlength-mono* **by** (*metis add.commute nat-add-left-cancel-le nlength-Suc-le plus-1-eq-Suc trans-le-add2*)
moreover have *H > 0*
using *H* **by** *simp*
ultimately have *nlength idx + nlength H ≤ Suc (nlength (Suc idx) + nlength H) ∗ H*
by (*metis* (*no-types, opaque-lifting*) *Suc-le-eq Suc-neq-Zero mult.assoc mult.commute mult-eq-1-iff mult-le-mono nat-mult-eq-cancel-disj*)
then show *?thesis*
by *simp*
qed
also have ... *= 36 + 110 ∗ (Suc (nlength (Suc idx) + nlength H) ∗ H)*^{2} *+ Suc H ∗ (43 + 9 ∗ (nlength (Suc idx) + nlength H)) +*
3 ∗ nlength H + 2 ∗ (nlength idx + nlength H)
by *simp*
also have ... *≤ 36 + 110 ∗ (Suc (nlength (Suc idx) + nlength H) ∗ H)*^{2} *+ Suc H ∗ (43 + 9 ∗ (nlength (Suc idx) + nlength H)) +*
3 ∗ (Suc (nlength (Suc idx) + nlength H) ∗ H)^2 + 2 ∗ (nlength idx + nlength H)
proof −
have *nlength H ≤ Suc (nlength (Suc idx) + nlength H) ∗ H*
using *H* **by** (*simp add*: *nlength-le-n trans-le-add1*)
then have *nlength H ≤ (Suc (nlength (Suc idx) + nlength H) ∗ H)^2*
by (*meson le-refl le-trans power2-nat-le-imp-le*)
then show *?thesis*
by *simp*
qed
also have ... *= 36 + 113 ∗ (Suc (nlength (Suc idx) + nlength H) ∗ H)*^{2} *+ Suc H ∗ (43 + 9 ∗ (nlength (Suc idx) + nlength H)) +*
2 ∗ (nlength idx + nlength H)
by *simp*
also have ... *≤ 36 + 113 ∗ (Suc (nlength (Suc idx) + nlength H) ∗ H)*^{2} *+ Suc H ∗ (43 + 9 ∗ (nlength (Suc idx) + nlength H)) +*
2 ∗ (Suc (nlength (Suc idx) + nlength H) ∗ H) ^ 2
proof −
have *nlength idx + nlength H ≤ Suc (nlength (Suc idx) + nlength H)*
using *nlength-mono* **by** (*simp add*: *le-SucI*)
also have ... *≤ Suc (nlength (Suc idx) + nlength H) ∗ H*
using *H* **by** (*metis Suc-eq-plus1 le-add2 mult.commute mult-le-mono1 nat-mult-1 numeral-eq-Suc order-trans*)

also have ... ≤ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*)^*2*
by (*simp add*: *power2-eq-square*)
finally have *nlength idx* + *nlength H* ≤ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*)^*2* **.**
then show *?thesis*
by *simp*
qed
also have ... = *79* + *115* ∗ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*)^2 +
9 ∗ (*nlength* (*Suc idx*) + *nlength H*) + (*43* + *9* ∗ (*nlength* (*Suc idx*) + *nlength H*)) ∗ *H*
by *simp*
also have ... = *79* + *115* ∗ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*)^2 +
9 ∗ (*nlength* (*Suc idx*) + *nlength H*) + *43* ∗ *H* + *9* ∗ (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*
by *algebra*
also have ... ≤ *79* + *115* ∗ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*)^2 +
9 ∗ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*) ^ *2* + *43* ∗ *H* + *9* ∗ (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*
proof −
have *nlength* (*Suc idx*) + *nlength H* ≤ *Suc* (*nlength* (*Suc idx*) + *nlength H*)
by *simp*
also have ... ≤ *Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*
using *H*
by (*metis One-nat-def add-leD1 le-refl mult-le-mono mult-numeral-1-right numeral-3-eq-3 numeral-nat*(*1*) *plus-1-eq-Suc*)
also have ... ≤ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*) ^ *2*
by (*simp add*: *power2-eq-square*)
finally have *nlength* (*Suc idx*) + *nlength H* ≤ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*) ^ *2* **.**
then show *?thesis*
by *simp*
qed
also have ... = *79* + *124* ∗ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*)^2 + *43* ∗ *H* + *9* ∗ (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*
by *simp*
also have ... ≤ *79* + *124* ∗ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*)^2 +
43 ∗ *H* + *9* ∗ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*) ^ *2*
proof −
have (*nlength* (*Suc idx*) + *nlength H*) ∗ *H* ≤ *Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*
by *simp*
then have (*nlength* (*Suc idx*) + *nlength H*) ∗ *H* ≤ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*) ^ *2*
by (*metis nat-le-linear power2-nat-le-imp-le verit-la-disequality*)
then show *?thesis*
by *linarith*
qed
also have ... = *79* + *133* ∗ (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*)^2 + *43* ∗ *H*
by *simp*
also have ... ≤ *79* + *133* ∗ (*9*∗*H*^*3* ∗ (*nlength* (*Suc idx*))^*2* + *4*∗*H*^*4*) + *43* ∗ *H*
proof −
let *?m* = *nlength* (*Suc idx*)
let *?l* = *Suc ?m*
have (*Suc* (*nlength* (*Suc idx*) + *nlength H*) ∗ *H*)^2 = ((*?l* + *nlength H*) ∗ *H*)^2
by *simp*
also have ... = (*?l*∗*H* + *nlength H*∗*H*) ^ *2*
by *algebra*
also have ... ≤ (*?l*∗*H* + *H*∗*H*) ^ *2*
using *nlength-le-n* **by** *simp*
also have ... = (*?l*∗*H*)^*2* + *2*∗*?l*∗*H*^*3* + *H*^*4*
by *algebra*
also have ... ≤ (*?l*∗*H*)^*2* + *2*∗*?l*^*2*∗*H*^*3* + *H*^*4*
by (*metis Suc-n-not-le-n add-le-mono1 mult-le-mono1 mult-le-mono2 nat-add-left-cancel-le not-less-eq-eq power2-nat-le-imp-le*)
also have ... = *?l*^*2*∗(*H*^*2* + *2*∗*H*^*3*) + *H*^*4*
by *algebra*
also have ... ≤ *?l*^*2*∗(*H*^*3* + *2*∗*H*^*3*) + *H*^*4*
proof −
have *H*^*2* ≤ *H*^*3*
using *pow-mono* **by** (*simp add*: *numeral-3-eq-3 numerals*(*2*))

then show *?thesis*
by *simp*
qed
also have ... = $?l\hat{}2*3*H\hat{}3 + H\hat{}4$
by *simp*
also have ... = $(?m\hat{}2 + 2 * ?m + 1)*3*H\hat{}3 + H\hat{}4$
by (*smt* (*verit*) *add.commute add-Suc mult-2 nat-1-add-1 one-power2 plus-1-eq-Suc power2-sum*)
also have ... ≤ $(?m\hat{}2 + 2 * ?m\hat{}2 + 1)*3*H\hat{}3 + H\hat{}4$
using *linear-le-pow* **by** *simp*
also have ... = $(3*?m\hat{}2 + 1)*3*H\hat{}3 + H\hat{}4$
by *simp*
also have ... = $9*?m\hat{}2*H\hat{}3 + 3*H\hat{}3 + H\hat{}4$
by *algebra*
also have ... ≤ $9*?m\hat{}2*H\hat{}3 + 3*H\hat{}4 + H\hat{}4$
using *pow-mono'* **by** *simp*
also have ... = $9*H\hat{}3 * ?m\hat{}2 + 4*H\hat{}4$
by *simp*
finally have $(Suc\ (nlength\ (Suc\ idx) + nlength\ H) * H)^2 \le 9*H\hat{}3 * ?m\hat{}2 + 4*H\hat{}4$ **.**
then show *?thesis*
by *simp*
qed
also have ... = $79 + 133 * 9*H\hat{}3 * (nlength\ (Suc\ idx))\hat{}2 + 133*4*H\hat{}4 + 43 * H$
by *simp*
also have ... ≤ $79 + 133 * 9*H\hat{}3 * (nlength\ (Suc\ idx))\hat{}2 + 133*4*H\hat{}4 + 43 * H\hat{}4$
using *linear-le-pow* **by** *simp*
also have ... ≤ $79*H\hat{}4 + 133 * 9*H\hat{}3 * (nlength\ (Suc\ idx))\hat{}2 + 133*4*H\hat{}4 + 43 * H\hat{}4$
using *H* **by** *simp*
also have ... = $654 * H\hat{}4 + 1197 * H\hat{}3 * (nlength\ (Suc\ idx))\hat{}2$
by *simp*
also have ... ≤ $654 * H\hat{}4 + 1197 * H\hat{}4 * (nlength\ (Suc\ idx))\hat{}2$
using *pow-mono'* **by** *simp*
also have ... ≤ $654 * H\hat{}4 * (nlength\ (Suc\ idx))\hat{}2 + 1197 * H\hat{}4 * (nlength\ (Suc\ idx))\hat{}2$
using *nlength-mono nlength-1-simp*
by (*metis add-le-mono1 le-add1 mult-le-mono2 mult-numeral-1-right numerals*(*1*) *one-le-power plus-1-eq-Suc*)
also have ... = $1851 * H\hat{}4 * (nlength\ (Suc\ idx))\hat{}2$
by *simp*
finally have $?ttt \le 1851 * H\hat{}4 * (nlength\ (Suc\ idx))\hat{}2$ **.**
then show *?thesis*
using *assms tm5 transforms-monotone tps5'-eq-tps5* **by** *simp*
qed

end

end

lemma *transforms-tm-PsigammaI* [*transforms-intros*]:
fixes j :: *tapeidx*
fixes $tps\ tps'$ :: *tape list* **and** $ttt\ H\ k\ idx\ \kappa$ :: *nat*
assumes $length\ tps = k$ **and** $0 < j$ **and** $j + 7 < k$
and $H \ge 3$
and $\kappa \le H$
assumes
$tps\ !\ j = (\lfloor idx \rfloor_N, 1)$
$tps\ !\ (j + 1) = (\lfloor H \rfloor_N, 1)$
$tps\ !\ (j + 2) = (\lfloor \kappa \rfloor_N, 1)$
$tps\ !\ (j + 3) = (\lfloor [] \rfloor, 1)$
$tps\ !\ (j + 4) = (\lfloor [] \rfloor, 1)$
$tps\ !\ (j + 5) = (\lfloor [] \rfloor, 1)$
$tps\ !\ (j + 6) = (\lfloor [] \rfloor, 1)$
$tps\ !\ (j + 7) = (\lfloor [] \rfloor, 1)$
assumes $ttt = 1851 * H\hat{}4 * (nlength\ (Suc\ idx))\hat{}2$
assumes $tps' = tps$
$[j + 6 := (\lfloor nll\text{-}Psi\ (idx * H)\ H\ \kappa \rfloor_{NLL}, 1)]$

shows *transforms* (*tm-Psigamma j*) *tps ttt tps'*
proof –
interpret *loc*: *turing-machine-Psigamma j* **.**
show *?thesis*
using *loc.tm5' loc.tps5'-def loc.tm5-eq-tm-Psigamma assms* **by** *simp*
qed

7.5 A Turing machine for Υ formulas

The CNF formula Φ_7 is made of CNF formulas $\Upsilon(\gamma_i)$ with $\gamma_i = [i \cdot H, (i + 1) \cdot H)$. In this section we build a Turing machine that outputs such CNF formulas.

7.5.1 A Turing machine for singleton clauses

The Υ formulas, just like the Ψ formulas, are conjunctions of singleton clauses. The next Turing machine outputs singleton clauses. The Turing machine has two parameters: a Boolean *incr* and a tape index j. It receives a variable index on tape j, a CNF formula as list of lists of numbers on tape $j + 2$ and a number H on tape $j + 3$. The TM appends to the formula on tape $j + 2$ a singleton clause with a positive or negative (depending on *incr*) literal derived from the variable index. It also decrements H and increments the variable index, which makes it more suitable for use in a loop constructing an Υ formula.

Given our encoding of literals, what the TM actually does is doubling the number on tape $j + 1$ and optionally (if *incr* is true) incrementing it.

definition *tm-times2-appendl* :: *bool* ⇒ *tapeidx* ⇒ *machine* **where**
tm-times2-appendl incr j ≡
tm-copyn j (*j* + *1*) ;;
tm-times2 (*j* + *1*) ;;
(*if incr then tm-incr* (*j* + *1*) *else* []) ;;
tm-to-list (*j* + *1*) ;;
tm-appendl (*j* + *2*) (*j* + *1*) ;;
tm-erase-cr (*j* + *1*) ;;
tm-incr j ;;
tm-decr (*j* + *3*)

lemma *tm-times2-appendl-tm*:
assumes *0* < *j* **and** *j* + *3* < *k* **and** *G* ≥ *6*
shows *turing-machine k G* (*tm-times2-appendl incr j*)
unfolding *tm-times2-appendl-def*
using *Nil-tm tm-incr-tm tm-to-list-tm tm-appendl-tm tm-decr-tm tm-erase-cr-tm tm-times2-tm assms tm-copyn-tm*
by *simp*

locale *turing-machine-times2-appendl* =
fixes *j* :: *tapeidx*
begin

context
fixes *tps0* :: *tape list* **and** *v H k* :: *nat* **and** *nss* :: *nat list list* **and** *incr* :: *bool*
assumes *jk*: *length tps0* = *k 0* < *j j* + *3* < *k*
assumes *tps0*:
tps0 ! *j* = (⌊*v*⌋$_N$, *1*)
tps0 ! (*j* + *1*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *2*) = *nlltape nss*
tps0 ! (*j* + *3*) = (⌊*H*⌋$_N$, *1*)
begin

definition *tm1* ≡ *tm-copyn j* (*j* + *1*)
definition *tm2* ≡ *tm1* ;; *tm-times2* (*j* + *1*)
definition *tm3* ≡ *tm2* ;; (*if incr then tm-incr* (*j* + *1*) *else* [])
definition *tm4* ≡ *tm3* ;; *tm-to-list* (*j* + *1*)
definition *tm5* ≡ *tm4* ;; *tm-appendl* (*j* + *2*) (*j* + *1*)
definition *tm6* ≡ *tm5* ;; *tm-erase-cr* (*j* + *1*)
definition *tm7* ≡ *tm6* ;; *tm-incr j*

definition *tm8* ≡ *tm7* ;; *tm-decr* (*j* + *3*)

lemma *tm8-eq-tm-times2appendl*: *tm8* ≡ *tm-times2-appendl incr j*
  using *tm8-def tm7-def tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def tm-times2-appendl-def* **by** *simp*

definition *tps1* ≡ *tps0*
  [*j* + *1* := (⌊*v*⌋$_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
  assumes *ttt* = *14* + *3* ∗ *nlength v*
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def*
proof (*tform tps*: *tps1-def tps0 jk*)
  show *tps0* ! (*j* + *1*) = (⌊*0*⌋$_N$, *1*)
    using *jk tps0 canrepr-0* **by** *simp*
  show *ttt* = *14* + *3* ∗ (*nlength v* + *nlength 0*)
    using *assms* **by** *simp*
qed

definition *tps2* ≡ *tps0*
  [*j* + *1* := (⌊*2* ∗ *v*⌋$_N$, *1*)]

lemma *tm2* [*transforms-intros*]:
  assumes *ttt* = *19* + *5* ∗ *nlength v*
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def* **by** (*tform tps*: *tps2-def tps1-def jk assms*)

definition *tps3* ≡ *tps0*
  [*j* + *1* := (⌊*2* ∗ *v* + (*if incr then 1 else 0*)⌋$_N$, *1*)]

lemma *tm3-True*:
  assumes *ttt* = *24* + *5* ∗ *nlength v* + *2* ∗ *nlength* (*2* ∗ *v*) **and** *incr*
  shows *transforms tm3 tps0 ttt tps3*
  unfolding *tm3-def*
proof (*tform tps*: *tps3-def tps2-def jk*)
  let *?t* = *5* + *2* ∗ *nlength* (*2* ∗ *v*)
  have *transforms* (*tm-incr* (*j* + *1*)) *tps2 ?t tps3*
    by (*tform tps*: *tps3-def tps2-def jk assms*(*2*))
  then show *transforms* (*if incr then tm-incr* (*j* + *1*) *else* []) *tps2 ?t tps3*
    using *assms*(*2*) **by** *simp*
  show *ttt* = *19* + *5* ∗ *nlength v* + *?t*
    using *assms* **by** *simp*
qed

lemma *tm3-False*:
  assumes *ttt* = *19* + *5* ∗ *nlength v* **and** ¬ *incr*
  shows *transforms tm3 tps0 ttt tps3*
  unfolding *tm3-def*
proof (*tform tps*: *tps3-def tps2-def jk assms*)
  show *transforms* (*if incr then tm-incr* (*j* + *1*) *else* []) *tps2 0 tps3*
    using *transforms-Nil jk tps3-def tps2-def assms*(*2*) **by** *simp*
qed

lemma *tm3*:
  assumes *ttt* = *24* + *5* ∗ *nlength v* + *2* ∗ *nlength* (*2* ∗ *v*)
  shows *transforms tm3 tps0 ttt tps3*
  using *tm3-True tm3-False assms transforms-monotone* **by** (*cases incr*) *simp-all*

lemma *tm3'* [*transforms-intros*]:
  assumes *ttt* = *26* + *7* ∗ *nlength v*
  shows *transforms tm3 tps0 ttt tps3*
proof −
  have *nlength* (*2* ∗ *v*) ≤ *Suc* (*nlength v*)

using *nlength-times2* **by** *simp*
then show *?thesis*
using *assms tm3 transforms-monotone* **by** *simp*
qed

definition *tps4* ≡ *tps0*
$[j + 1 := (\lfloor[2 * v + (if\ incr\ then\ 1\ else\ 0)]\rfloor_{NL},\ 1)]$

lemma *tm4*:
assumes $ttt = 31 + 7 * nlength\ v + 2 * nlength\ (2 * v + (if\ incr\ then\ 1\ else\ 0))$
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def* **by** (*tform tps*: *tps4-def tps3-def jk assms*)

lemma *tm4'* [*transforms-intros*]:
assumes $ttt = 33 + 9 * nlength\ v$
shows *transforms tm4 tps0 ttt tps4*
proof −
have $nlength\ (2 * v + (if\ incr\ then\ 1\ else\ 0)) \le Suc\ (nlength\ v)$
using *nlength-times2 nlength-times2plus1* **by** *simp*
then show *?thesis*
using *assms tm4 transforms-monotone* **by** *simp*
qed

definition *tps5* ≡ *tps0*
$[j + 1 := (\lfloor[2 * v + (if\ incr\ then\ 1\ else\ 0)]\rfloor_{NL},\ 1),$
$j + 2 := nlltape\ (nss\ @\ [[2 * v + (if\ incr\ then\ 1\ else\ 0)]])]$

lemma *tm5* [*transforms-intros*]:
assumes $ttt = 39 + 9 * nlength\ v + 2 * nllength\ [2 * v + (if\ incr\ then\ 1\ else\ 0)]$
shows *transforms tm5 tps0 ttt tps5*
unfolding *tm5-def*
proof (*tform tps*: *tps5-def tps4-def jk tps0*)
show $ttt = 33 + 9 * nlength\ v + (7 + nlllength\ nss - Suc\ (nlllength\ nss) +$
$2 * nllength\ [2 * v + (if\ incr\ then\ 1\ else\ 0)])$
using *assms* **by** *simp*
qed

definition *tps6* ≡ *tps0*
$[j + 2 := nlltape\ (nss\ @\ [[2 * v + (if\ incr\ then\ 1\ else\ 0)]])]$

lemma *tm6*:
assumes $ttt = 46 + 9 * nlength\ v + 4 * nllength\ [2 * v + (if\ incr\ then\ 1\ else\ 0)]$
shows *transforms tm6 tps0 ttt tps6*
unfolding *tm6-def*
proof (*tform tps*: *tps6-def tps5-def jk*)
let $?zs = numlist\ [2 * v + (if\ incr\ then\ 1\ else\ 0)]$
show $tps5\ :::\ (j + 1) = \lfloor ?zs \rfloor$
using *jk tps5-def nlcontents-def* **by** *simp*
show *proper-symbols ?zs*
using *proper-symbols-numlist* **by** *simp*
show $tps6 = tps5[j + 1 := (\lfloor[]\rfloor,\ 1)]$
using *jk tps6-def tps5-def tps0*
by (*metis* (*no-types*, *lifting*) *add-left-cancel list-update-id list-update-overwrite list-update-swap numeral-eq-one-iff semiring-norm*(*83*))
show $ttt = 39 + 9 * nlength\ v + 2 * nllength\ [2 * v + (if\ incr\ then\ 1\ else\ 0)] +$
$(tps5\ :\#:\ (j + 1) + 2 * length\ (numlist\ [2 * v + (if\ incr\ then\ 1\ else\ 0)]) + 6)$
using *assms nllength-def tps5-def jk* **by** *simp*
qed

lemma *tm6'* [*transforms-intros*]:
assumes $ttt = 54 + 13 * nlength\ v$
shows *transforms tm6 tps0 ttt tps6*
proof −

have *nlength* (*2* ∗ *v* + (*if incr then 1 else 0*)) ≤ *Suc* (*nlength v*)
  using *nlength-times2 nlength-times2plus1* **by** *simp*
then show *?thesis*
  using *assms tm6 transforms-monotone nllength* **by** *simp*
qed

definition *tps7* ≡ *tps0*
[*j* := (⌊*Suc v*⌋$_N$, *1*),
 j + *2* := *nlltape* (*nss* @ [[*2* ∗ *v* + (*if incr then 1 else 0*)]])]

lemma *tm7* [*transforms-intros*]:
  assumes *ttt* = *59* + *15* ∗ *nlength v*
  shows *transforms tm7 tps0 ttt tps7*
  unfolding *tm7-def* **by** (*tform tps*: *tps7-def tps6-def tps0 jk assms*)

definition *tps8* ≡ *tps0*
[*j* := (⌊*Suc v*⌋$_N$, *1*),
 j + *2* := *nlltape* (*nss* @ [[*2* ∗ *v* + (*if incr then 1 else 0*)]]),
 j + *3* := (⌊*H* − *1*⌋$_N$, *1*)]

lemma *tm8*:
  assumes *ttt* = *67* + *15* ∗ *nlength v* + *2* ∗ *nlength H*
  shows *transforms tm8 tps0 ttt tps8*
  unfolding *tm8-def* **by** (*tform tps*: *tps8-def tps0 tps7-def jk time*: *assms*)

end

end

lemma *transforms-tm-times2-appendlI* [*transforms-intros*]:
  fixes *j* :: *tapeidx* **and** *incr* :: *bool*
  fixes *tps tps'* :: *tape list* **and** *ttt v H k* :: *nat* **and** *nss* :: *nat list list*
  assumes *length tps* = *k* **and** *0* < *j* **and** *j* + *3* < *k*
  assumes
    tps ! *j* = (⌊*v*⌋$_N$, *1*)
    tps ! (*j* + *1*) = (⌊[]⌋, *1*)
    tps ! (*j* + *2*) = *nlltape nss*
    tps ! (*j* + *3*) = (⌊*H*⌋$_N$, *1*)
  assumes *ttt* = *67* + *15* ∗ *nlength v* + *2* ∗ *nlength H*
  assumes *tps'* = *tps*
    [*j* := (⌊*Suc v*⌋$_N$, *1*),
     j + *2* := *nlltape* (*nss* @ [[*2* ∗ *v* + (*if incr then 1 else 0*)]]),
     j + *3* := (⌊*H* − *1*⌋$_N$, *1*)]
  shows *transforms* (*tm-times2-appendl incr j*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-times2-appendl j* **.**
  show *?thesis*
    using *assms loc.tm8 loc.tps8-def loc.tm8-eq-tm-times2appendl* **by** *metis*
qed

7.5.2 A Turing machine for $\Upsilon(\gamma_i)$ formulas

We will not need the general Υ formulas, but only $\Upsilon(\gamma_i)$ for $\gamma_i = [i \cdot H, (i+1) \cdot H)$. Represented as list of lists of numbers they look like this (for $H \geq 3$):

definition *nll-Upsilon* :: *nat* ⇒ *nat* ⇒ *nat list list* **where**
  nll-Upsilon idx len ≡ [[*2* ∗ (*idx* ∗ *len*) + *1*], [*2* ∗ (*idx* ∗ *len* + *1*) + *1*]] @ *map* (λ*i*. [*2* ∗ (*idx* ∗ *len* + *i*)]) [*3*..<*len*]

lemma *nll-Upsilon*:
  assumes *len* ≥ *3*
  shows *nll-Upsilon idx len* = *formula-n* (Υ [*idx*∗*len*..<*idx*∗*len*+*len*])
    (**is** *?lhs* = *?rhs*)
proof (*rule nth-equalityI*)

show *len*: *length ?lhs* = *length ?rhs*
  using *nll-Upsilon-def Upsilon-def formula-n-def assms* **by** *simp*
have *length ?lhs* = *len* − *1*
  using *nll-Upsilon-def assms* **by** *simp*
define *nss* **where** *nss* = [[*2* ∗ (*idx* ∗ *len*) + *1*], [*2* ∗ (*idx* ∗ *len* + *1*) + *1*]]
then have ∗: *?lhs* = *nss* @ *map* (λ*i*. [*2* ∗ (*idx* ∗ *len* + *i*)]) [*3*..<*len*]
  using *nll-Upsilon-def* **by** *simp*
have *length nss* = *2*
  using *nss-def* **by** *simp*
let *?ups* = Υ [*idx*∗*len*..<*idx*∗*len*+*len*]
show *?lhs* ! *i* = *?rhs* ! *i* **if** *i* < *length ?lhs* **for** *i*
proof (*cases i* < *2*)
  case *True*
  then have *?lhs* ! *i* = *nss* ! *i*
    using ∗ ‹*length nss* = *2*› **by** (*simp add*: *nth-append*)
  then have *lhs*: *?lhs* ! *i* = [*2* ∗ (*idx* ∗ *len* + *i*) + *1*]
    using *nss-def True* **by** (*cases i* = *0*) *auto*

  have *?ups* ! *i* = [*Pos* (*idx*∗*len*+*i*)]
    unfolding *Upsilon-def* **using** *True assms* **by** (*cases i* = *0*) *auto*
  moreover have *?rhs* ! *i* = *clause-n* (*?ups* ! *i*)
    using *that len formula-n-def* **by** *simp*
  ultimately have *?rhs* ! *i* = *clause-n* [*Pos* (*idx*∗*len*+*i*)]
    by *simp*
  then have *?rhs* ! *i* = [*Suc* (*2*∗(*idx*∗*len*+*i*))]
    using *clause-n-def* **by** *simp*
  then show *?thesis*
    using *lhs* **by** *simp*
next
  case *False*
  then have *?lhs* ! *i* = *map* (λ*i*. [*2* ∗ (*idx* ∗ *len* + *i*)]) [*3*..<*len*] ! (*i* − *2*)
    using ‹*length nss* = *2*› ∗ **by** (*simp add*: *nth-append*)
  also have ... = (λ*i*. [*2* ∗ (*idx* ∗ *len* + *i*)]) ([*3*..<*len*] ! (*i* − *2*))
    using *False that* ‹*length nss* = *2*› ∗ **by** *simp*
  also have ... = (λ*i*. [*2* ∗ (*idx* ∗ *len* + *i*)]) (*i* + *1*)
    using *False that* ‹*length nss* = *2*› ∗ **by** *simp*
  also have ... = [*2* ∗ (*idx* ∗ *len* + *i* + *1*)]
    by *simp*
  finally have *lhs*: *?lhs* ! *i* = [*2* ∗ (*idx* ∗ *len* + *i* + *1*)] **.**

  have *?ups* ! *i* = *map* (λ*s*. [*Neg s*]) (*drop 3* [*idx*∗*len*..<*idx*∗*len*+*len*]) ! (*i* − *2*)
    using *Upsilon-def False that formula-n-def len* **by** *auto*
  also have ... = (λ*s*. [*Neg s*]) (*drop 3* [*idx*∗*len*..<*idx*∗*len*+*len*] ! (*i* − *2*))
    using *Upsilon-def False that formula-n-def len* **by** *auto*
  also have ... = (λ*s*. [*Neg s*]) (*idx* ∗ *len* + *i* + *1*)
    using *Upsilon-def False that formula-n-def len* **by** *auto*
  finally have *?ups* ! *i* = [*Neg* (*idx* ∗ *len* + *i* + *1*)] **.**
  moreover have *?rhs* ! *i* = *clause-n* (*?ups* ! *i*)
    using *Upsilon-def False that formula-n-def len* **by** *auto*
  ultimately have *?rhs* ! *i* = *clause-n* [*Neg* (*idx* ∗ *len* + *i* + *1*)]
    by *simp*
  then show *?thesis*
    using *clause-n-def lhs* **by** *simp*
qed
qed

lemma *nlllength-nll-Upsilon-le*:
  assumes *len* ≥ *3*
  shows *nlllength* (*nll-Upsilon idx len*) ≤ *len* ∗ (*4* + *nlength idx* + *nlength len*)
proof −
  define *f* :: *nat* ⇒ *nat list* **where** *f* = (λ*i*. [*2* ∗ (*idx* ∗ *len* + *i*)])
  let *?nss* = *map f* [*3*..<*len*]
  have *nlllength ?nss* = (∑*ns*←*?nss*. *Suc* (*nllength ns*))

using *nllength f-def* **by** *simp*
also have ... = $(\sum i{\leftarrow}[3..<len].\ (\lambda ns.\ Suc\ (nllength\ ns))\ (f\ i))$
by (*metis* (*no-types, lifting*) *map-eq-conv map-map o-apply*)
also have ... = $(\sum i{\leftarrow}[3..<len].\ Suc\ (nllength\ ([2 * (idx * len + i)])))$
using *f-def* **by** *simp*
also have ... = $(\sum i{\leftarrow}[3..<len].\ Suc\ (Suc\ (nlength\ (2 * (idx * len + i)))))$
using *nllength* **by** *simp*
also have ... $\leq (\sum i{\leftarrow}[3..<len].\ Suc\ (Suc\ (nlength\ (2 * (Suc\ idx * len)))))$
using *nlength-mono*
sum-list-mono[*of* [*3..<len*]
$\lambda i.\ Suc\ (Suc\ (nlength\ (2 * (idx * len + i))))$
$\lambda i.\ Suc\ (Suc\ (nlength\ (2 * (Suc\ idx * len))))$]
by *simp*
also have ... = $Suc\ (Suc\ (nlength\ (2 * (Suc\ idx * len)))) * (len - 3)$
using *assms sum-list-const*[*of* - [*3..<len*]] **by** *simp*
also have ... $\leq Suc\ (Suc\ (Suc\ (nlength\ (Suc\ idx * len)))) * (len - 3)$
using *nlength-times2 Suc-le-mono mult-le-mono1* **by** *presburger*
also have ... = $(len - 3) * (3 + nlength\ (Suc\ idx * len))$
by (*simp add*: *Suc3-eq-add-3*)
finally have $*$: *nlllength ?nss* $\leq (len - 3) * (3 + nlength\ (Suc\ idx * len))$ **.**

let *?nss2* = $[[2 * (idx * len) + 1], [2 * (idx * len + 1) + 1]]$
have *nlllength ?nss2* = $(\sum ns{\leftarrow}?nss2.\ Suc\ (nllength\ ns))$
using *nlllength* **by** *simp*
also have ... = $Suc\ (nllength\ [2 * (idx * len) + 1]) + Suc\ (nllength\ [2 * (idx * len + 1) + 1])$
by *simp*
also have ... = $Suc\ (Suc\ (nlength\ (2 * (idx * len) + 1))) + Suc\ (Suc\ (nlength\ (2 * (idx * len + 1) + 1)))$
using *nllength* **by** *simp*
also have ... $\leq Suc\ (Suc\ (nlength\ (2 * (Suc\ idx * len)))) + Suc\ (Suc\ (nlength\ (2 * (idx * len + 1) + 1)))$
using *nlength-mono assms* **by** *simp*
also have ... $\leq Suc\ (Suc\ (nlength\ (2 * (Suc\ idx * len)))) + Suc\ (Suc\ (nlength\ (2 * (Suc\ idx * len))))$
using *nlength-mono assms* **by** *simp*
also have ... = $2 * Suc\ (Suc\ (nlength\ (2 * (Suc\ idx * len))))$
by *simp*
also have ... $\leq 2 * Suc\ (Suc\ (Suc\ (nlength\ (Suc\ idx * len))))$
using *nlength-times2* **by** (*meson Suc-le-mono mult-le-mono nle-le*)
also have ... = $2 * (3 + nlength\ (Suc\ idx * len))$
by *simp*
finally have $**$: *nlllength ?nss2* $\leq 2 * (3 + nlength\ (Suc\ idx * len))$ **.**

have *nll-Upsilon idx len* = *?nss2* @ *?nss*
using *nll-Upsilon-def f-def* **by** *simp*
then have *nlllength* (*nll-Upsilon idx len*) = *nlllength ?nss2* + *nlllength ?nss*
by (*metis length-append nlllength-def numlistlist-append*)
then have *nlllength* (*nll-Upsilon idx len*) $\leq 2 * (3 + nlength\ (Suc\ idx * len)) + (len - 3) * (3 + nlength\ (Suc\ idx * len))$
using $*$ $**$ **by** *simp*
also have ... = $(2 + (len - 3)) * (3 + nlength\ (Suc\ idx * len))$
by *simp*
also have ... = $(len - 1) * (3 + nlength\ (Suc\ idx * len))$
using *assms Nat.le-imp-diff-is-add* **by** *fastforce*
also have ... $\leq len * (3 + nlength\ (Suc\ idx * len))$
by *simp*
also have ... $\leq len * (3 + nlength\ (Suc\ idx) + nlength\ len)$
using *nlength-prod* **by** (*metis ab-semigroup-add-class.add-ac*(*1*) *mult-le-mono2 nat-add-left-cancel-le*)
also have ... $\leq len * (4 + nlength\ idx + nlength\ len)$
using *nlength-Suc* **by** *simp*
finally show *?thesis* **.**
qed

The next Turing machine outputs CNF formulas of the shape $\Upsilon(\gamma_i)$, where $\gamma_i = [i \cdot H, (i+1) \cdot H)$. It expects a number i on tape j and a number H on tape $j+1$. It writes a representation of the formula to tape $j+4$.

definition *tm-Upsilongamma* :: *tapeidx* ⇒ *machine* **where**
tm-Upsilongamma j ≡
tm-copyn (*j* + *1*) (*j* + *5*) ;;
tm-mult j (*j* + *1*) (*j* + *2*) ;;
tm-times2-appendl True (*j* + *2*) ;;
tm-times2-appendl True (*j* + *2*) ;;
tm-decr (*j* + *5*) ;;
tm-incr (*j* + *2*) ;;
WHILE [] ; *λrs. rs* ! (*j* + *5*) ≠ □ *DO*
tm-times2-appendl False (*j* + *2*)
DONE ;;
tm-erase-cr (*j* + *2*) ;;
tm-cr (*j* + *4*)

lemma *tm-Upsilongamma-tm*:
assumes $0 < j$ **and** $j + 5 < k$ **and** $G \geq 6$
shows *turing-machine k G* (*tm-Upsilongamma j*)
unfolding *tm-Upsilongamma-def*
using *tm-copyn-tm Nil-tm tm-decr-tm tm-times2-appendl-tm tm-decr-tm tm-mult-tm tm-incr-tm assms turing-machine-loop-turing-machine tm-erase-cr-tm tm-cr-tm*
by *simp*

locale *turing-machine-Upsilongamma* =
fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-copyn* (*j* + *1*) (*j* + *5*)
definition *tm2* ≡ *tm1* ;; *tm-mult j* (*j* + *1*) (*j* + *2*)
definition *tm3* ≡ *tm2* ;; *tm-times2-appendl True* (*j* + *2*)
definition *tm4* ≡ *tm3* ;; *tm-times2-appendl True* (*j* + *2*)
definition *tm5* ≡ *tm4* ;; *tm-decr* (*j* + *5*)
definition *tm6* ≡ *tm5* ;; *tm-incr* (*j* + *2*)
definition *tmB* ≡ *tm-times2-appendl False* (*j* + *2*)
definition *tmL* ≡ *WHILE* [] ; *λrs. rs* ! (*j* + *5*) ≠ □ *DO tmB DONE*
definition *tm7* ≡ *tm6* ;; *tmL*
definition *tm8* ≡ *tm7* ;; *tm-erase-cr* (*j* + *2*)
definition *tm9* ≡ *tm8* ;; *tm-cr* (*j* + *4*)

lemma *tm9-eq-tm-Upsilongamma*: *tm9* = *tm-Upsilongamma j*
using *tm9-def tm8-def tm7-def tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def tmB-def tmL-def tm-Upsilongamma-def*
by *simp*

context
fixes *tps0* :: *tape list* **and** *idx H k* :: *nat*
assumes *jk*: *length tps0* = *k* $0 < j$ $j + 5 < k$
and *H*: $H \geq 3$
assumes *tps0*:
tps0 ! *j* = ($\lfloor idx \rfloor_N$, *1*)
tps0 ! (*j* + *1*) = ($\lfloor H \rfloor_N$, *1*)
tps0 ! (*j* + *2*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *3*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *4*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *5*) = (⌊[]⌋, *1*)
begin

definition *tps1* ≡ *tps0*
[*j* + *5* := ($\lfloor H \rfloor_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *14* + *3* ∗ *nlength H*
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
proof (*tform tps*: *tps1-def tps0 jk*)

show *tps0* ! $(j + 5) = (\lfloor 0 \rfloor_N, 1)$
using *jk tps0 canrepr-0* **by** *simp*
show $ttt = 14 + 3 * (nlength\ H + nlength\ 0)$
using *assms* **by** *simp*
qed

definition $tps2 \equiv tps0$
$[j + 5 := (\lfloor H \rfloor_N, 1),$
$j + 2 := (\lfloor idx * H \rfloor_N, 1)]$

lemma *tm2* [*transforms-intros*]:
assumes $ttt = 18 + 3 * nlength\ H + 26 * (nlength\ idx + nlength\ H)\hat{}2$
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def*
proof (*tform tps*: *tps2-def tps1-def jk tps0*)
show *tps1* ! $(j + 2) = (\lfloor 0 \rfloor_N, 1)$
using *tps1-def jk canrepr-1 tps0*
by (*metis add-left-imp-eq canrepr-0 nth-list-update-neq′ numeral-eq-iff semiring-norm*(*89*))
show $ttt = 14 + 3 * nlength\ H + (4 + 26 * (nlength\ idx + nlength\ H) * (nlength\ idx + nlength\ H))$
using *assms* **by** *algebra*
qed

definition $tps3 \equiv tps0$
$[j + 5 := (\lfloor H - 1 \rfloor_N, 1),$
$j + 4 := nlltape\ ([[2 * (idx * H) + 1]]),$
$j + 2 := (\lfloor idx * H + 1 \rfloor_N, 1)]$

lemma *tm3* [*transforms-intros*]:
assumes $ttt = 85 + 5 * nlength\ H + 26 * (nlength\ idx + nlength\ H)\hat{}2 + 15 * nlength\ (idx * H)$
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def*
proof (*tform tps*: *tps3-def tps2-def jk tps0*)
have $*$: $j + 2 + 1 = j + 3\ j + 2 + 2 = j + 4\ j + 2 + 3 = j + 5$
by *simp-all*
show *tps2* ! $(j + 2 + 1) = (\lfloor [] \rfloor, 1)$
using *jk tps2-def tps0* **by** (*simp only*: $*$) *simp*
show *tps2* ! $(j + 2 + 2) = nlltape\ []$
using *jk tps2-def tps0 nllcontents-Nil* **by** (*simp only*: $*$) *simp*
show *tps2* ! $(j + 2 + 3) = (\lfloor H \rfloor_N, 1)$
using *jk tps2-def tps0* **by** (*simp only*: $*$) *simp*
show $tps3 = tps2$
$[j + 2 := (\lfloor Suc\ (idx * H) \rfloor_N, 1),$
$j + 2 + 2 := nlltape\ ([]\ @\ [[2 * (idx * H) + (if\ True\ then\ 1\ else\ 0)]]),$
$j + 2 + 3 := (\lfloor H - 1 \rfloor_N, 1)]$
unfolding *tps3-def tps2-def*
by (*simp only*: $*$) (*simp add*: *list-update-swap*[*of Suc* (*Suc j*)] *list-update-swap-less*[*of j+4*])
show $ttt = 18 + 3 * nlength\ H + 26 * (nlength\ idx + nlength\ H)^2 +$
$(67 + 15 * nlength\ (idx * H) + 2 * nlength\ H)$
using *assms* **by** *simp*
qed

definition $tps4 \equiv tps0$
$[j + 5 := (\lfloor H - 2 \rfloor_N, 1),$
$j + 4 := nlltape\ ([[2 * (idx * H) + 1], [2 * (idx * H + 1) + 1]]),$
$j + 2 := (\lfloor idx * H + 2 \rfloor_N, 1)]$

lemma *tm4* [*transforms-intros*]:
assumes $ttt = 152 + 5 * nlength\ H + 26 * (nlength\ idx + nlength\ H)\hat{}2 + 15 * nlength\ (idx * H) +$
$15 * nlength\ (Suc\ (idx * H)) + 2 * nlength\ (H - 1)$
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform tps*: *tps4-def tps3-def jk tps0*)
have $*$: $j + 2 + 1 = j + 3\ j + 2 + 2 = j + 4\ j + 2 + 3 = j + 5$

by *simp-all*
show *tps3* ! $(j + 2 + 1) = (\lfloor[]\rfloor, 1)$
using *jk tps3-def tps0* **by** (*simp only*: $*$) *simp*
show *tps3* ! $(j + 2 + 2) =$ *nlltape* $[[2 * (idx * H) + 1]]$
using *jk tps3-def tps0* **by** (*simp only*: $*$) *simp*
show *tps3* ! $(j + 2 + 3) = (\lfloor H - 1\rfloor_N, 1)$
using *jk tps3-def tps0* **by** (*simp only*: $*$) *simp*
have *2*: $2 = Suc\ (Suc\ 0)$
by *simp*
show *tps4* = *tps3*
$[j + 2 := (\lfloor Suc\ (Suc\ (idx * H))\rfloor_N, 1),$
$j + 2 + 2 := nlltape\ ([[2 * (idx * H) + 1]]\ @\ [[2 * Suc\ (idx * H) + (if\ True\ then\ 1\ else\ 0)]]),$
$j + 2 + 3 := (\lfloor H - 1 - 1\rfloor_N, 1)]$
unfolding *tps4-def tps3-def* **by** (*simp only*: $*$) (*simp add*: *2 list-update-swap*)
show $ttt = 85 + 5 * nlength\ H + 26 * (nlength\ idx + nlength\ H)^2 + 15 * nlength\ (idx * H) +$
$(67 + 15 * nlength\ (Suc\ (idx * H)) + 2 * nlength\ (H - 1))$
using *assms* **by** *simp*
qed

definition *tps5* $\equiv$ *tps0*
$[j + 5 := (\lfloor H - 3\rfloor_N, 1),$
$j + 4 := nlltape\ ([[2 * (idx * H) + 1], [2 * (idx * H + 1) + 1]]),$
$j + 2 := (\lfloor idx * H + 2\rfloor_N, 1)]$

lemma *tm5* [*transforms-intros*]:
assumes $ttt = 160 + 5 * nlength\ H + 26 * (nlength\ idx + nlength\ H)\hat{}2 + 15 * nlength\ (idx * H) +$
$15 * nlength\ (Suc\ (idx * H)) + 2 * nlength\ (H - 1) + 2 * nlength\ (H - 2)$
shows *transforms tm5 tps0 ttt tps5*
unfolding *tm5-def*
proof (*tform tps*: *tps5-def tps4-def jk tps0*)
show $ttt = 152 + 5 * nlength\ H + 26 * (nlength\ idx + nlength\ H)^2 + 15 * nlength\ (idx * H) +$
$15 * nlength\ (Suc\ (idx * H)) + 2 * nlength\ (H - 1) + (8 + 2 * nlength\ (H - 2))$
using *assms* **by** *simp*
qed

definition *tps6* $\equiv$ *tps0*
$[j + 5 := (\lfloor H - 3\rfloor_N, 1),$
$j + 4 := nlltape\ ([[2 * (idx * H) + 1], [2 * (idx * H + 1) + 1]]),$
$j + 2 := (\lfloor idx * H + 3\rfloor_N, 1)]$

lemma *tm6*:
assumes $ttt = 165 + 5 * nlength\ H + 26 * (nlength\ idx + nlength\ H)\hat{}2 + 15 * nlength\ (idx * H) +$
$15 * nlength\ (Suc\ (idx * H)) + 2 * nlength\ (H - 1) + 2 * nlength\ (H - 2) +$
$2 * nlength\ (Suc\ (Suc\ (idx * H)))$
shows *transforms tm6 tps0 ttt tps6*
unfolding *tm6-def*
proof (*tform tps*: *tps6-def tps5-def jk tps0*)
show $tps6 = tps5[j + 2 := (\lfloor Suc\ (Suc\ (Suc\ (idx * H)))\rfloor_N, 1)]$
unfolding *tps5-def tps6-def*
by (*simp only*: *One-nat-def Suc-1 add-2-eq-Suc′ add-Suc-right numeral-3-eq-3*) (*simp add*: *list-update-swap*)
show $ttt = 160 + 5 * nlength\ H + 26 * (nlength\ idx + nlength\ H)^2 + 15 * nlength\ (idx * H) +$
$15 * nlength\ (Suc\ (idx * H)) + 2 * nlength\ (H - 1) + 2 * nlength\ (H - 2) +$
$(5 + 2 * nlength\ (Suc\ (Suc\ (idx * H))))$
using *assms* **by** *simp*
qed

lemma *tm6′* [*transforms-intros*]:
assumes $ttt = 165 + 41 * nlength\ (Suc\ idx * H) + 26 * (nlength\ idx + nlength\ H)\hat{}2$
shows *transforms tm6 tps0 ttt tps6*
proof –
let $?ttt = 165 + 5 * nlength\ H + 26 * (nlength\ idx + nlength\ H)\hat{}2 + 15 * nlength\ (idx * H) +$
$15 * nlength\ (Suc\ (idx * H)) + 2 * nlength\ (H - 1) + 2 * nlength\ (H - 2) +$
$2 * nlength\ (Suc\ (Suc\ (idx * H)))$

have *?ttt* $\leq$ *165 + 5 ∗ nlength (Suc idx ∗ H) + 26 ∗ (nlength idx + nlength H)^2 + 15 ∗ nlength (idx ∗ H) +*
15 ∗ nlength (Suc (idx ∗ H)) + 2 ∗ nlength (H − 1) + 2 ∗ nlength (H − 2) +
2 ∗ nlength (Suc (Suc (idx ∗ H)))
using *nlength-mono* **by** *simp*
also have ... $\leq$ *165 + 5 ∗ nlength (Suc idx ∗ H) + 26 ∗ (nlength idx + nlength H)^2 + 15 ∗ nlength (idx ∗ H) +*
15 ∗ nlength (Suc (idx ∗ H)) + 2 ∗ nlength (Suc idx ∗ H) + 2 ∗ nlength (H − 2) +
2 ∗ nlength (Suc (Suc (idx ∗ H)))
using *nlength-mono* **by** *simp*
also have ... $\leq$ *165 + 5 ∗ nlength (Suc idx ∗ H) + 26 ∗ (nlength idx + nlength H)^2 + 15 ∗ nlength (idx ∗ H) +*
15 ∗ nlength (Suc (idx ∗ H)) + 2 ∗ nlength (Suc idx ∗ H) + 2 ∗ nlength (Suc idx ∗ H) +
2 ∗ nlength (Suc (Suc (idx ∗ H)))
using *nlength-mono* **by** *simp*
also have ... $\leq$ *165 + 5 ∗ nlength (Suc idx ∗ H) + 26 ∗ (nlength idx + nlength H)^2 + 15 ∗ nlength (Suc idx ∗ H) +*
15 ∗ nlength (Suc (idx ∗ H)) + 2 ∗ nlength (Suc idx ∗ H) + 2 ∗ nlength (Suc idx ∗ H) +
2 ∗ nlength (Suc (Suc (idx ∗ H)))
using *nlength-mono* **by** *simp*
also have ... $\leq$ *165 + 5 ∗ nlength (Suc idx ∗ H) + 26 ∗ (nlength idx + nlength H)^2 + 15 ∗ nlength (Suc idx ∗ H) +*
15 ∗ nlength (Suc idx ∗ H) + 2 ∗ nlength (Suc idx ∗ H) + 2 ∗ nlength (Suc idx ∗ H) +
2 ∗ nlength (Suc (Suc (idx ∗ H)))
using *nlength-mono H* **by** *simp*
also have ... $\leq$ *165 + 5 ∗ nlength (Suc idx ∗ H) + 26 ∗ (nlength idx + nlength H)^2 + 15 ∗ nlength (Suc idx ∗ H) +*
15 ∗ nlength (Suc idx ∗ H) + 2 ∗ nlength (Suc idx ∗ H) + 2 ∗ nlength (Suc idx ∗ H) +
2 ∗ nlength (Suc idx ∗ H)
using *nlength-mono H* **by** *simp*
also have ... = *165 + 41 ∗ nlength (Suc idx ∗ H) + 26 ∗ (nlength idx + nlength H)^2*
using *nlength-mono* **by** *simp*
finally have *?ttt* $\leq$ *ttt*
using *assms* **by** *simp*
then show *?thesis*
using *tm6 transforms-monotone* **by** *simp*
qed

definition *tpsL t* ≡ *tps0*
[j + 5 := (⌊H − 3 − t⌋$_N$, 1),
j + 4 := nlltape ([[2 ∗ (idx ∗ H) + 1], [2 ∗ (idx ∗ H + 1) + 1]] @ map (λi. [2 ∗ (idx ∗ H + i)]) [3..<3 + t]),
j + 2 := (⌊idx ∗ H + 3 + t⌋$_N$, 1)]

lemma *tpsL-eq-tps6*: *tpsL 0 = tps6*
using *tpsL-def tps6-def* **by** *simp*

lemma *map-Suc-append*: *a ≤ b ⟹ map f [a..<Suc b] = map f [a..<b] @ [f b]*
by *simp*

lemma *tmB*:
assumes *ttt = 67 + 15 ∗ nlength (idx ∗ H + 3 + t) + 2 ∗ nlength (H − 3 − t)*
shows *transforms tmB (tpsL t) ttt (tpsL (Suc t))*
unfolding *tmB-def*
proof (*tform tps*: *tpsL-def jk tps0*)
have ∗: *j + 2 + 1 = j + 3 j + 2 + 2 = j + 4 j + 2 + 3 = j + 5*
by *simp-all*
show *tpsL t ! (j + 2 + 1) = (⌊[]⌋, 1)*
using *jk tpsL-def tps0* **by** (*simp only*: ∗) *simp*
let *?nss = [[2 ∗ (idx ∗ H) + 1], [2 ∗ (idx ∗ H + 1) + 1]] @ map (λi. [2 ∗ (idx ∗ H + i)]) [3..<3 + t]*
show *tpsL t ! (j + 2 + 2) = nlltape ?nss*
using *jk tpsL-def* **by** (*simp only*: ∗) *simp*
show *tpsL t ! (j + 2 + 3) = (⌊H − 3 − t⌋$_N$, 1)*

using *jk tpsL-def* **by** (*simp only*: ∗) *simp*
have $map\ (\lambda i.\ [2 * (idx * H + i)])\ [3..<3 + Suc\ t] =$
$map\ (\lambda i.\ [2 * (idx * H + i)])\ [3..<3 + t]\ @\ [[2 * (idx * H + 3 + t) + (if\ False\ then\ 1\ else\ 0)]]$
using *map-Suc-append*[*of 3 3 + t* $\lambda i.\ [2 * (idx * H + i)]$] **by** *simp*
then have $[[2 * (idx * H) + 1], [2 * (idx * H + 1) + 1]]\ @\ map\ (\lambda i.\ [2 * (idx * H + i)])\ [3..<3 + Suc\ t] =$
$?nss\ @\ [[2 * (idx * H + 3 + t) + (if\ False\ then\ 1\ else\ 0)]]$
by *simp*
then show $tpsL\ (Suc\ t) = (tpsL\ t)$
$[j + 2 := (\lfloor Suc\ (idx * H + 3 + t)\rfloor_N,\ 1),$
$j + 2 + 2 := nlltape\ (?nss\ @\ [[2 * (idx * H + 3 + t) + (if\ False\ then\ 1\ else\ 0)]]),$
$j + 2 + 3 := (\lfloor H - 3 - t - 1\rfloor_N,\ 1)]$
unfolding *tpsL-def*
by (*simp only*: ∗) (*simp add*: *list-update-swap*[*of Suc* (*Suc j*)] *list-update-swap-less*[*of j* + *4*])
show $ttt = 67 + 15 * nlength\ (idx * H + 3 + t) + 2 * nlength\ (H - 3 - t)$
using *assms* **by** *simp*
qed

lemma *tmB′* [*transforms-intros*]:
assumes $ttt = 67 + 15 * nlength\ (Suc\ idx * H) + 2 * nlength\ H$
and $t < H - 3$
shows *transforms tmB* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
proof −
let $?ttt = 67 + 15 * nlength\ (idx * H + 3 + t) + 2 * nlength\ (H - 3 - t)$
have $?ttt \le 67 + 15 * nlength\ (idx * H + 3 + t) + 2 * nlength\ H$
using *nlength-mono* **by** *simp*
also have ... $\le 67 + 15 * nlength\ (Suc\ idx * H) + 2 * nlength\ H$
using *assms*(*2*) *nlength-mono* **by** *simp*
finally have $?ttt \le ttt$
using *assms*(*1*) **by** *simp*
then show *?thesis*
using *tmB transforms-monotone* **by** *blast*
qed

lemma *tmL*:
assumes $ttt = H * (70 + 17 * nlength\ (Suc\ idx * H))$
shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsL* (*H* − *3*))
unfolding *tmL-def*
proof (*tform*)
have $read\ (tpsL\ t)\ !\ (j + 5) = \Box \longleftrightarrow H - 3 - t = 0$ **for** *t*
using *jk tpsL-def read-ncontents-eq-0* **by** *simp*
then show $\bigwedge t.\ t < H - 3 \Longrightarrow read\ (tpsL\ t)\ !\ (j + 5) \ne \Box$
and $\neg\ read\ (tpsL\ (H - 3))\ !\ (j + 5) \ne \Box$
by *simp-all*
show $(H - 3) * (67 + 15 * nlength\ (Suc\ idx * H) + 2 * nlength\ H + 2) + 1 \le ttt$
(**is** $?lhs \le ttt$)
proof −
have $?lhs = (H - 3) * (69 + 15 * nlength\ (Suc\ idx * H) + 2 * nlength\ H) + 1$
by *simp*
also have ... $\le (H - 3) * (69 + 15 * nlength\ (Suc\ idx * H) + 2 * nlength\ (Suc\ idx * H)) + 1$
using *nlength-mono* **by** *simp*
also have ... $= (H - 3) * (69 + 17 * nlength\ (Suc\ idx * H)) + 1$
by *simp*
also have ... $\le H * (69 + 17 * nlength\ (Suc\ idx * H)) + 1$
by *simp*
also have ... $\le H * (69 + 17 * nlength\ (Suc\ idx * H)) + H$
using *H* **by** *simp*
also have ... $= H * (70 + 17 * nlength\ (Suc\ idx * H))$
by *algebra*
finally show $?lhs \le ttt$
using *assms* **by** *simp*
qed
qed

definition *tps7* ≡ *tps0*
  [*j* + *4* := *nlltape* ([[*2* ∗ (*idx* ∗ *H*) + *1*], [*2* ∗ (*idx* ∗ *H* + *1*) + *1*]] @ *map* (λ*i*. [*2* ∗ (*idx* ∗ *H* + *i*)]) [*3*..<*H*]),
  [*j* + *5* := (⌊*0*⌋$_N$, *1*),
   j + *4* := *nlltape* ([[*2* ∗ (*idx* ∗ *H*) + *1*], [*2* ∗ (*idx* ∗ *H* + *1*) + *1*]] @ *map* (λ*i*. [*2* ∗ (*idx* ∗ *H* + *i*)]) [*3*..<*H*]),
   j + *2* := (⌊*Suc idx* ∗ *H*⌋$_N$, *1*)]

lemma *tpsL-eq-tps7*: *tpsL* (*H* − *3*) = *tps7*
proof −
  let *?t* = *H* − *3*
  have (⌊*H* − *3* − *?t*⌋$_N$, *1*) = (⌊*0*⌋$_N$, *1*)
    by *simp*
  moreover have *nlltape* ([[*2* ∗ (*idx* ∗ *H*) + *1*], [*2* ∗ (*idx* ∗ *H* + *1*) + *1*]] @ *map* (λ*i*. [*2* ∗ (*idx* ∗ *H* + *i*)]) [*3*..<*3* + *?t*]) =
      nlltape ([[*2* ∗ (*idx* ∗ *H*) + *1*], [*2* ∗ (*idx* ∗ *H* + *1*) + *1*]] @ *map* (λ*i*. [*2* ∗ (*idx* ∗ *H* + *i*)]) [*3*..<*H*])
    using *H* **by** *simp*
  moreover have (⌊*idx* ∗ *H* + *3* + *?t*⌋$_N$, *1*) = (⌊*Suc idx* ∗ *H*⌋$_N$, *1*)
    using *H* **by** (*simp add*: *add.commute*)
  ultimately show *?thesis*
    using *tpsL-def tps7-def* **by** *simp*
qed

lemma *tmL′* [*transforms-intros*]:
  assumes *ttt* = *H* ∗ (*70* + *17* ∗ *nlength* (*Suc idx* ∗ *H*))
  shows *transforms tmL tps6 ttt tps7*
  using *tmL tpsL-eq-tps6 tpsL-eq-tps7 assms* **by** *simp*

lemma *tm7* [*transforms-intros*]:
  assumes *ttt* = *165* + *41* ∗ *nlength* (*H* + *idx* ∗ *H*) + *26* ∗ (*nlength idx* + *nlength H*)^2 +
      H ∗ (*70* + *17* ∗ *nlength* (*H* + *idx* ∗ *H*))
  shows *transforms tm7 tps0 ttt tps7*
  unfolding *tm7-def* **by** (*tform tps*: *tps7-def tpsL-def jk tps0 time*: *assms*)

definition *tps8* ≡ *tps0*
  [*j* + *5* := (⌊*0*⌋$_N$, *1*),
   j + *4* := *nlltape* ([[*2* ∗ (*idx* ∗ *H*) + *1*], [*2* ∗ (*idx* ∗ *H* + *1*) + *1*]] @ *map* (λ*i*. [*2* ∗ (*idx* ∗ *H* + *i*)]) [*3*..<*H*]),
   j + *2* := (⌊[]⌋, *1*)]

lemma *tm8*:
  assumes *ttt* = *172* + *43* ∗ *nlength* (*H* + *idx* ∗ *H*) + *26* ∗ (*nlength idx* + *nlength H*)^2 +
      H ∗ (*70* + *17* ∗ *nlength* (*H* + *idx* ∗ *H*))
  shows *transforms tm8 tps0 ttt tps8*
  unfolding *tm8-def*
proof (*tform tps*: *tps8-def tps7-def jk tps0 assms*)
  show *proper-symbols* (*canrepr* (*H* + *idx* ∗ *H*))
    using *proper-symbols-canrepr* **by** *simp*
qed

definition *tps8′* ≡ *tps0*[*j* + *4* := *nlltape* (*nll-Upsilon idx H*)]

lemma *tps8′-eq-tps8*: *tps8′* = *tps8*
proof −
  define *tps* **where** *tps* = *tps0*
    [*j* + *4* := *nlltape* ([[*2* ∗ (*idx* ∗ *H*) + *1*], [*2* ∗ (*idx* ∗ *H* + *1*) + *1*]] @ *map* (λ*i*. [*2* ∗ (*idx* ∗ *H* + *i*)]) [*3*..<*H*])]
  then have *tps* = *tps8*
    using *jk tps8-def canrepr-0 tps0*
    by (*smt* (*verit*, *best*) *add-left-imp-eq arith-simps*(*4*) *list-update-id list-update-swap num.simps*(*2*) *numeral-eq-iff semiring-norm*(*83*))
  then show *?thesis*
    using *nll-Upsilon-def tps8′-def tps-def* **by** *simp*
qed

lemma *tm8′* [*transforms-intros*]:
  assumes *ttt* = *199* ∗ *H* ∗ (*nlength idx* + *nlength H*)^*2*
  shows *transforms tm8 tps0 ttt tps8′*

proof −
  let *?ttt* = *172* + *43* ∗ *nlength* (*H* + *idx* ∗ *H*) + *26* ∗ (*nlength idx* + *nlength H*)^2 +
      H ∗ (*70* + *17* ∗ *nlength* (*H* + *idx* ∗ *H*))
  have *?ttt* = *172* + *43* ∗ *nlength* (*H* + *idx* ∗ *H*) + *26* ∗ (*nlength idx* + *nlength H*)^2 +
      70 ∗ *H* + *17* ∗ *H* ∗ *nlength* (*H* + *idx* ∗ *H*)
    by *algebra*
  also have ... = *172* + *70* ∗ *H* + (*17* ∗ *H* + *43*) ∗ *nlength* (*H* + *idx* ∗ *H*) + *26* ∗ (*nlength idx* + *nlength H*)^2
    by *algebra*
  also have ... = *172* + *70* ∗ *H* + (*17* ∗ *H* + *43*) ∗ *nlength* (*Suc idx* ∗ *H*) + *26* ∗ (*nlength idx* + *nlength H*)^2
    by *simp*
  also have ... ≤ *172* + *70* ∗ *H* + (*17* ∗ *H* + *43*) ∗ (*nlength* (*Suc idx*) + *nlength H*) + *26* ∗ (*nlength idx* + *nlength H*)^2
    using *nlength-prod* **by** (*meson add-le-mono mult-le-mono order-refl*)
  also have ... ≤ *172* + *70* ∗ *H* + (*17* ∗ *H* + *43*) ∗ (*Suc* (*nlength idx*) + *nlength H*) + *26* ∗ (*nlength idx* + *nlength H*)^2
    using *nlength-Suc add-le-mono le-refl mult-le-mono* **by** *presburger*
  also have ... = *172* + *70* ∗ *H* + (*17* ∗ *H* + *43*) + (*17* ∗ *H* + *43*) ∗ (*nlength idx* + *nlength H*) + *26* ∗ (*nlength idx* + *nlength H*)^2
    by *simp*
  also have ... = *215* + *87* ∗ *H* + (*17* ∗ *H* + *43*) ∗ (*nlength idx* + *nlength H*) + *26* ∗ (*nlength idx* + *nlength H*)^2
    by *simp*
  also have ... ≤ *215* + *87* ∗ *H* + (*17* ∗ *H* + *43*) ∗ (*nlength idx* + *nlength H*)ˆ*2* + *26* ∗ (*nlength idx* + *nlength H*)^2
    using *linear-le-pow* **by** *simp*
  also have ... = *215* + *87* ∗ *H* + (*17* ∗ *H* + *69*) ∗ (*nlength idx* + *nlength H*)ˆ*2*
    by *algebra*
  also have ... ≤ *159* ∗ *H* + (*17* ∗ *H* + *69*) ∗ (*nlength idx* + *nlength H*)ˆ*2*
    using *H* **by** *simp*
  also have ... ≤ *159* ∗ *H* + *40* ∗ *H* ∗ (*nlength idx* + *nlength H*)ˆ*2*
    using *H* **by** *simp*
  also have ... ≤ *199* ∗ *H* ∗ (*nlength idx* + *nlength H*)ˆ*2*
  proof −
    have *nlength H* > *0*
      using *H nlength-0* **by** *simp*
    then have *nlength idx* + *nlength H* ≥ *1*
      by *linarith*
    then show *?thesis*
      by *simp*
  qed
  finally have *?ttt* ≤ *ttt*
    using *assms* **by** *simp*
  then show *?thesis*
    using *tps8′-eq-tps8 tm8 transforms-monotone* **by** *simp*
qed

definition *tps9* ≡ *tps0*
  [*j* + *4* := (⌊*nll-Upsilon idx H*⌋$_{NLL}$, *1*)]

lemma *tm9*:
  assumes *ttt* = *199* ∗ *H* ∗ (*nlength idx* + *nlength H*)ˆ*2* + *Suc* (*Suc* (*Suc* (*nlllength* (*nll-Upsilon idx H*))))
  shows *transforms tm9 tps0 ttt tps9*
  unfolding *tm9-def*
proof (*tform tps*: *tps8′-def tps9-def jk tps0 assms*)
  show *clean-tape* (*tps8′* ! (*j* + *4*))
    using *tps8′-def jk tps0 clean-tape-nllcontents* **by** *simp*
qed

lemma *tm9′* [*transforms-intros*]:
  assumes *ttt* = *205* ∗ *H* ∗ (*nlength idx* + *nlength H*)ˆ*2*
  shows *transforms tm9 tps0 ttt tps9*
proof −
  let *?ttt* = *199* ∗ *H* ∗ (*nlength idx* + *nlength H*)ˆ*2* + *Suc* (*Suc* (*Suc* (*nlllength* (*nll-Upsilon idx H*))))

```
  have ?ttt ≤ 199 * H * (nlength idx + nlength H)^2 + Suc (Suc (Suc (H * (4 + nlength idx + nlength H))))
    using nlllength-nll-Upsilon-le H by simp
  also have ... = 199 * H * (nlength idx + nlength H)^2 + 3 + H * (4 + nlength idx + nlength H)
    by simp
  also have ... = 199 * H * (nlength idx + nlength H)^2 + 3 + 4 * H + H * (nlength idx + nlength H)
    by algebra
  also have ... ≤ 199 * H * (nlength idx + nlength H)^2 + 5 * H + H * (nlength idx + nlength H)
    using H by simp
  also have ... ≤ 199 * H * (nlength idx + nlength H)^2 + 5 * H + H * (nlength idx + nlength H)^2
    using linear-le-pow by simp
  also have ... = 200 * H * (nlength idx + nlength H)^2 + 5 * H
    by simp
  also have ... ≤ 205 * H * (nlength idx + nlength H)^2
  proof −
    have nlength H ≥ 1
      using H nlength-0 by (metis less-one not-less not-numeral-le-zero)
    then show ?thesis
      by simp
  qed
  finally have ?ttt ≤ ttt
    using assms by simp
  then show ?thesis
    using tm9 transforms-monotone by simp
qed

end

end

lemma transforms-tm-UpsilongammaI [transforms-intros]:
  fixes j :: tapeidx
  fixes tps tps' :: tape list and ttt idx H k :: nat
  assumes length tps = k and 0 < j and j + 5 < k
    and H ≥ 3
  assumes
    tps ! j = (⌊idx⌋_N, 1)
    tps ! (j + 1) = (⌊H⌋_N, 1)
    tps ! (j + 2) = (⌊[]⌋, 1)
    tps ! (j + 3) = (⌊[]⌋, 1)
    tps ! (j + 4) = (⌊[]⌋, 1)
    tps ! (j + 5) = (⌊[]⌋, 1)
  assumes ttt = 205 * H * (nlength idx + nlength H)^2
  assumes tps' = tps[j + 4 := (⌊nll-Upsilon idx H⌋_NLL, 1)]
  shows transforms (tm-Upsilongamma j) tps ttt tps'
proof −
  interpret loc: turing-machine-Upsilongamma j .
  show ?thesis
    using assms loc.tm9-eq-tm-Upsilongamma loc.tm9' loc.tps9-def by simp
qed

end
```

7.6 Turing machines for the parts of Φ

```
theory Sat-TM-CNF
  imports Aux-TM-Reducing
begin
```

In this section we build Turing machines for all parts $\Phi_0, \ldots, \Phi_9$ of the CNF formula Φ. Some of them (Φ_0, Φ_1, Φ_2, and Φ_8) are just fixed-length sequences of Ψ formulas constructible by fixed-length sequences of *tm-Psigamma* machines. Others (Φ_3, Φ_4, Φ_5, Φ_6) are variable-length and require looping over a *tm-Psigamma* machine. The TM for Φ_7 is a loop over *tm-Upsilongamma*. Lastly, the TM for Φ_9 is a loop over a TM that generates the formulas χ_t.

Ideally we would want to prove the semantics of the TMs inside the locale *reduction-sat*, in which Φ was defined. However, we use locales to prove the semantics of TMs, and locales cannot be nested. For this reason we have to define the TMs on the theory level and prove their semantics there, too, just as we have done with all TMs until now. In the next chapter the semantics lemmas will be transferred to the locale *reduction-sat*.

Unlike most TMs so far, the TMs in this section are not meant to be reusable but serve a special purpose, namely to be combined into one large TM computing Φ. For this reason the TMs are somewhat peculiar. For example, they write their output to the fixed tape 1 rather than having a parameter for the output tape. They also often expect the tapes to be initialized in a very special way. Moreover, the TMs often leave the work tapes in a "dirty" state with remnants of intermediate calculations. The combined TM for all of Φ will simply allocate a new batch of work tapes for every individual TM.

7.6.1 A Turing machine for Φ_0

The next Turing machine expects a number i on tape j and a number H on tape $j+1$ and outputs to tape 1 the formula $\Psi([i\dots H,(i{+}1)\dots H),1)\wedge\Psi([(i{+}1)\dots H,(i{+}2)\dots H),1)\wedge\Psi([(i{+}2)\dots H,(i{+}3)\dots H),0)$, which is just Φ_0 for suitable values of i and H.

definition *tm-PHI0* :: *tapeidx* ⇒ *machine* **where**
  tm-PHI0 j ≡
    tm-setn (*j* + *2*) *1* ;;
    tm-Psigamma j ;;
    tm-extendl-erase 1 (*j* + *6*) ;;
    tm-incr j ;;
    tm-Psigamma j ;;
    tm-extendl-erase 1 (*j* + *6*) ;;
    tm-incr j ;;
    tm-setn (*j* + *2*) *0* ;;
    tm-Psigamma j ;;
    tm-extendl 1 (*j* + *6*)

lemma *tm-PHI0-tm*:
  assumes *0* < *j* **and** *j* + *8* < *k* **and** *G* ≥ *6*
  shows *turing-machine k G* (*tm-PHI0 j*)
  unfolding *tm-PHI0-def*
  using *assms tm-Psigamma-tm tm-extendl-tm tm-erase-cr-tm tm-times2-tm tm-incr-tm tm-setn-tm tm-cr-tm tm-extendl-erase-tm*
  by *simp*

locale *turing-machine-PHI0* =
  fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-setn* (*j* + *2*) *1*
definition *tm2* ≡ *tm1* ;; *tm-Psigamma j*
definition *tm3* ≡ *tm2* ;; *tm-extendl-erase 1* (*j* + *6*)
definition *tm5* ≡ *tm3* ;; *tm-incr j*
definition *tm6* ≡ *tm5* ;; *tm-Psigamma j*
definition *tm7* ≡ *tm6* ;; *tm-extendl-erase 1* (*j* + *6*)
definition *tm9* ≡ *tm7* ;; *tm-incr j*
definition *tm10* ≡ *tm9* ;; *tm-setn* (*j* + *2*) *0*
definition *tm11* ≡ *tm10* ;; *tm-Psigamma j*
definition *tm12* ≡ *tm11* ;; *tm-extendl 1* (*j* + *6*)

lemma *tm12-eq-tm-PHI0*: *tm12* = *tm-PHI0 j*
  using *tm12-def tm11-def tm10-def tm9-def tm7-def tm6-def tm5-def*
  using *tm3-def tm2-def tm1-def tm-PHI0-def*
  by *simp*

context
  fixes *tps0* :: *tape list* **and** *k idx H* :: *nat*
  assumes *jk*: *length tps0* = *k 1* < *j j* + *8* < *k*

and *H*: $H \geq 3$
assumes *tps0*:
tps0 ! *1* = (⌊[]⌋, *1*)
tps0 ! *j* = (⌊*idx*⌋$_N$, *1*)
tps0 ! (*j* + *1*) = (⌊*H*⌋$_N$, *1*)
tps0 ! (*j* + *2*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *3*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *4*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *5*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *6*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *7*) = (⌊[]⌋, *1*)
tps0 ! (*j* + *8*) = (⌊[]⌋, *1*)
begin

definition *tps1* ≡ *tps0*
[*j* + *2* := (⌊*1*⌋$_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *12*
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
proof (*tform tps*: *tps0 tps1-def jk*)
show *tps0* ! (*j* + *2*) = (⌊*0*⌋$_N$, *1*)
using *tps0 jk canrepr-0* **by** *simp*
show *ttt* = *10* + *2* ∗ *nlength 0* + *2* ∗ *nlength 1*
using *assms canrepr-1* **by** *simp*
qed

definition *tps2* ≡ *tps0*
[*j* + *2* := (⌊*1*⌋$_N$, *1*),
j + *6* := (⌊*nll-Psi* (*idx* ∗ *H*) *H 1*⌋$_{NLL}$, *1*)]

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *12* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def* **by** (*tform tps*: *assms tps0 H tps1-def tps2-def jk*)

definition *tps3* ≡ *tps0*
[*j* + *2* := (⌊*1*⌋$_N$, *1*),
1 := *nlltape* (*nll-Psi* (*idx* ∗ *H*) *H* (*Suc 0*)),
j + *6* := (⌊[]⌋, *1*)]

lemma *tm3* [*transforms-intros*]:
assumes *ttt* = *23* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2 +
4 ∗ *nlllength* (*nll-Psi* (*idx* ∗ *H*) *H* (*Suc 0*))
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def*
proof (*tform tps*: *tps0 H tps2-def tps3-def jk time*: *assms*)
show *tps2* ! *1* = *nlltape* []
using *tps2-def jk nllcontents-Nil tps0* **by** *simp*
show *tps3* = *tps2*
[*1* := *nlltape* ([] @ *nll-Psi* (*idx* ∗ *H*) *H* (*Suc 0*)),
j + *6* := (⌊[]⌋, *1*)]
unfolding *tps3-def tps2-def* **using** *jk* **by** (*simp add*: *list-update-swap*)
qed

definition *tps5* ≡ *tps0*
[*j* + *2* := (⌊*1*⌋$_N$, *1*),
1 := *nlltape* (*nll-Psi* (*idx* ∗ *H*) *H* (*Suc 0*)),
j + *6* := (⌊[]⌋, *1*),
j := (⌊*Suc idx*⌋$_N$, *1*)]

lemma *tm5* [*transforms-intros*]:

assumes *ttt* = *28* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2 +
4 ∗ *nlllength* (*nll-Psi* (*idx* ∗ *H*) *H* *1*) +
2 ∗ *nlength idx*
shows *transforms tm5 tps0 ttt tps5*
unfolding *tm5-def* **by** (*tform tps*: *tps0 H tps3-def tps5-def jk time*: *assms*)

definition *tps6* ≡ *tps0*
[*j* := (⌊*Suc idx*⌋$_N$, *1*),
j + *2* := (⌊*1*⌋$_N$, *1*),
j + *6* := (⌊*nll-Psi* (*Suc idx* ∗ *H*) *H* (*Suc 0*)⌋$_{NLL}$, *1*),
1 := *nlltape* (*nll-Psi* (*idx* ∗ *H*) *H* *1*)]

lemma *tm6* [*transforms-intros*]:
assumes *ttt* = *28* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2 +
4 ∗ *nlllength* (*nll-Psi* (*idx* ∗ *H*) *H* *1*) + *2* ∗ *nlength idx* +
1851 ∗ *H* ^ *4* ∗ (*nlength* (*Suc* (*Suc idx*)))^2
shows *transforms tm6 tps0 ttt tps6*
unfolding *tm6-def*
proof (*tform tps*: *tps0 H tps5-def tps6-def jk time*: *assms*)
show *tps6* = *tps5*[*j* + *6* := (⌊*nll-Psi* (*Suc idx* ∗ *H*) *H* (*Suc 0*)⌋$_{NLL}$, *1*)]
unfolding *tps5-def tps6-def* **using** *jk*
by (*simp add*: *list-update-swap*[*of j*] *list-update-swap*[*of - j* + *6*])
qed

definition *tps7* ≡ *tps0*
[*j* := (⌊*Suc idx*⌋$_N$, *1*),
j + *2* := (⌊*1*⌋$_N$, *1*),
j + *6* := (⌊[]⌋, *1*),
1 := *nlltape* (*nll-Psi* (*idx* ∗ *H*) *H* *1* @ *nll-Psi* (*H* + *idx* ∗ *H*) *H* *1*)]

lemma *tm7* [*transforms-intros*]:
assumes *ttt* = *39* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2 +
4 ∗ *nlllength* (*nll-Psi* (*idx* ∗ *H*) *H* *1*) +
2 ∗ *nlength idx* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc* (*Suc idx*)))^2 +
4 ∗ *nlllength* (*nll-Psi* (*H* + *idx* ∗ *H*) *H* *1*)
shows *transforms tm7 tps0 ttt tps7*
unfolding *tm7-def* **by** (*tform tps*: *assms tps6-def tps7-def jk*)

definition *tps9* ≡ *tps0*
[*j* := (⌊*Suc* (*Suc idx*)⌋$_N$, *1*),
j + *2* := (⌊*1*⌋$_N$, *1*),
j + *6* := (⌊[]⌋, *1*),
1 := *nlltape* (*nll-Psi* (*idx* ∗ *H*) *H* *1* @ *nll-Psi* (*H* + *idx* ∗ *H*) *H* *1*)]

lemma *tm9* [*transforms-intros*]:
assumes *ttt* = *44* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2 +
4 ∗ *nlllength* (*nll-Psi* (*idx* ∗ *H*) *H* *1*) + *2* ∗ *nlength idx* +
1851 ∗ *H* ^ *4* ∗ (*nlength* (*Suc* (*Suc idx*)))^2 + *4* ∗ *nlllength* (*nll-Psi* (*H* + *idx* ∗ *H*) *H* *1*) +
2 ∗ *nlength* (*Suc idx*)
shows *transforms tm9 tps0 ttt tps9*
unfolding *tm9-def*
proof (*tform tps*: *tps0 H tps7-def tps9-def jk time*: *assms*)
show *tps9* = *tps7*[*j* := (⌊*Suc* (*Suc idx*)⌋$_N$, *1*)]
using *tps9-def tps7-def jk* **by** (*simp add*: *list-update-swap*)
qed

definition *tps10* ≡ *tps0*
[*j* := (⌊*Suc* (*Suc idx*)⌋$_N$, *1*),
j + *2* := (⌊*0*⌋$_N$, *1*),
j + *6* := (⌊[]⌋, *1*),
1 := *nlltape* (*nll-Psi* (*idx* ∗ *H*) *H* *1* @ *nll-Psi* (*H* + *idx* ∗ *H*) *H* *1*)]

lemma *tm10* [*transforms-intros*]:

assumes $ttt = 56 + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ idx))^2 +$
$4 * nlllength\ (nll\text{-}Psi\ (idx * H)\ H\ 1) + 2 * nlength\ idx +$
$1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ idx)))^2 + 4 * nlllength\ (nll\text{-}Psi\ (H + idx * H)\ H\ 1) +$
$2 * nlength\ (Suc\ idx)$
shows *transforms tm10 tps0 ttt tps10*
unfolding *tm10-def*
proof (*tform tps*: *tps0 H tps9-def tps10-def jk*)
show $ttt = 44 + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ idx))^2 +$
$4 * nlllength\ (nll\text{-}Psi\ (idx * H)\ H\ 1) + 2 * nlength\ idx +$
$1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ idx)))^2 + 4 * nlllength\ (nll\text{-}Psi\ (H + idx * H)\ H\ 1) +$
$2 * nlength\ (Suc\ idx) + (10 + 2 * nlength\ (Suc\ 0) + 2 * nlength\ 0)$
using *assms canrepr-1* **by** *simp*
qed

definition $tps11 \equiv tps0$
$[j := (\lfloor Suc\ (Suc\ idx)\rfloor_N,\ 1),$
$j + 2 := (\lfloor 0\rfloor_N,\ 1),$
$j + 6 := (\lfloor nll\text{-}Psi\ (Suc\ (Suc\ idx) * H)\ H\ 0\rfloor_{NLL},\ 1),$
$1 := nlltape\ (nll\text{-}Psi\ (idx * H)\ H\ 1\ @\ nll\text{-}Psi\ (H + idx * H)\ H\ 1)]$

lemma *tm11* [*transforms-intros*]:
assumes $ttt = 56 + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ idx))^2 +$
$4 * nlllength\ (nll\text{-}Psi\ (idx * H)\ H\ 1) + 2 * nlength\ idx +$
$1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ idx)))^2 + 4 * nlllength\ (nll\text{-}Psi\ (H + idx * H)\ H\ 1) +$
$2 * nlength\ (Suc\ idx) + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ (Suc\ idx))))^2$
shows *transforms tm11 tps0 ttt tps11*
unfolding *tm11-def* **by** (*tform tps*: *tps0 H tps10-def tps11-def jk time*: *assms*)

definition $tps12 \equiv tps0$
$[j := (\lfloor Suc\ (Suc\ idx)\rfloor_N,\ 1),$
$j + 2 := (\lfloor 0\rfloor_N,\ 1),$
$j + 6 := (\lfloor nll\text{-}Psi\ (Suc\ (Suc\ idx) * H)\ H\ 0\rfloor_{NLL},\ 1),$
$1 := nlltape\ (nll\text{-}Psi\ (idx * H)\ H\ 1\ @\ nll\text{-}Psi\ (H + idx * H)\ H\ 1\ @\ nll\text{-}Psi\ (Suc\ (Suc\ idx) * H)\ H\ 0)]$

lemma *tm12*:
assumes $ttt = 60 + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ idx))^2 +$
$4 * nlllength\ (nll\text{-}Psi\ (idx * H)\ H\ 1) + 2 * nlength\ idx +$
$1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ idx)))^2 + 4 * nlllength\ (nll\text{-}Psi\ (H + idx * H)\ H\ 1) +$
$2 * nlength\ (Suc\ idx) + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ (Suc\ idx))))^2 +$
$2 * nlllength\ (nll\text{-}Psi\ (H + (H + idx * H))\ H\ 0)$
shows *transforms tm12 tps0 ttt tps12*
unfolding *tm12-def* **by** (*tform tps*: *tps11-def tps12-def jk assms*)

lemma *tm12'*:
assumes $ttt = 5627 * H \mathbin{\hat{}} 4 * (3 + nlength\ (3 * H + idx * H))^2$
shows *transforms tm12 tps0 ttt tps12*
proof −
let $?ttt = 60 + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ idx))^2 +$
$4 * nlllength\ (nll\text{-}Psi\ (idx * H)\ H\ 1) + 2 * nlength\ idx +$
$1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ idx)))^2 + 4 * nlllength\ (nll\text{-}Psi\ (H + idx * H)\ H\ 1) +$
$2 * nlength\ (Suc\ idx) + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ (Suc\ idx))))^2 +$
$2 * nlllength\ (nll\text{-}Psi\ (H + (H + idx * H))\ H\ 0)$
have $?ttt \le 60 + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ idx))^2 +$
$4 * H * (3 + nlength\ (3 * H + idx * H)) + 2 * nlength\ idx +$
$1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ idx)))^2 + 4 * nlllength\ (nll\text{-}Psi\ (H + idx * H)\ H\ 1) +$
$2 * nlength\ (Suc\ idx) + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ (Suc\ idx))))^2 +$
$2 * nlllength\ (nll\text{-}Psi\ (H + (H + idx * H))\ H\ 0)$
using *nlllength-nll-Psi-le'*[*of idx* $*$ *H 2* $*$ *H* $+$ *idx* $*$ *H H*] **by** *simp*
also have $... \le 60 + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ idx))^2 +$
$4 * H * (3 + nlength\ (3 * H + idx * H)) + 2 * nlength\ idx +$
$1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ idx)))^2 + 4 * H * (3 + nlength\ (3 * H + idx * H)) +$
$2 * nlength\ (Suc\ idx) + 1851 * H \mathbin{\hat{}} 4 * (nlength\ (Suc\ (Suc\ (Suc\ idx))))^2 +$
$2 * nlllength\ (nll\text{-}Psi\ (H + (H + idx * H))\ H\ 0)$

using *nlllength-nll-Psi-le'*[*of H + idx ∗ H 2 ∗ H + idx ∗ H H*] **by** *simp*
also have ... $\leq$ *60 + 1851 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} +
4 ∗ H ∗ (3 + nlength (3 ∗ H + idx ∗ H)) + 2 ∗ nlength idx +
1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc idx)))^2 *+ 4 ∗ H ∗ (3 + nlength (3 ∗ H + idx ∗ H)) +*
2 ∗ nlength (Suc idx) + 1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc idx))))^2 +
2 ∗ H ∗ (3 + nlength (3 ∗ H + idx ∗ H))
using *nlllength-nll-Psi-le'*[*of H + (H + idx ∗ H) 2 ∗ H + idx ∗ H H*] **by** *simp*
also have ... = *60 + 1851 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} +
10 ∗ H ∗ (3 + nlength (3 ∗ H + idx ∗ H)) + 2 ∗ nlength idx +
1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc idx)))^2 *+ 2 ∗ nlength (Suc idx) +*
1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc idx))))^2
by *simp*
also have ... $\leq$ *60 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc idx)))*^{2} +
10 ∗ H ∗ (3 + nlength (3 ∗ H + idx ∗ H)) + 2 ∗ nlength idx +
1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc idx))))^2 *+ 2 ∗ nlength (Suc idx) +*
1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc idx))))^2
using *nlength-mono linear-le-pow* **by** *simp*
also have ... $\leq$ *60 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc idx))))*^{2} +
10 ∗ H ∗ (3 + nlength (3 ∗ H + idx ∗ H)) + 2 ∗ nlength idx +
1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc idx))))^2 *+ 2 ∗ nlength (Suc idx) +*
1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc idx))))^2
using *nlength-mono linear-le-pow* **by** *simp*
also have ... = *60 + 5553 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc idx))))*^{2} +
10 ∗ H ∗ (3 + nlength (3 ∗ H + idx ∗ H)) + 2 ∗ nlength idx + 2 ∗ nlength (Suc idx)
by *simp*
also have ... $\leq$ *60 + 5553 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc idx))))*^{2} +
10 ∗ H ∗ (3 + nlength (3 ∗ H + idx ∗ H)) + 2 ∗ nlength (Suc idx) + 2 ∗ nlength (Suc idx)
using *nlength-mono* **by** *simp*
also have ... = *60 + 5553 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc idx))))*^{2} +
10 ∗ H ∗ (3 + nlength (3 ∗ H + idx ∗ H)) + 4 ∗ nlength (Suc idx)
by *simp*
also have ... $\leq$ *60 + 5553 ∗ H ^ 4 ∗ (3 + nlength (3 ∗ H + idx ∗ H))*^{2} +
10 ∗ H ∗ (3 + nlength (3 ∗ H + idx ∗ H)) + 4 ∗ nlength (Suc idx)
proof −
have *Suc (Suc (Suc idx))* $\leq$ *3 ∗ H + idx ∗ H*
proof (*cases idx = 0*)
case *True*
then show *?thesis*
using *H* **by** *simp*
next
case *False*
then show *?thesis*
using *H*
by (*metis One-nat-def Suc3-eq-add-3 comm-semiring-class.distrib le-Suc-eq less-eq-nat.simps(1) mult.commute mult-1 mult-le-mono1 nle-le not-numeral-le-zero*)
qed
then have *nlength (Suc (Suc (Suc idx)))* $\leq$ *3 + nlength (3 ∗ H + idx ∗ H)*
using *nlength-mono trans-le-add2* **by** *presburger*
then have *nlength (Suc (Suc (Suc idx))) ^ 2* $\leq$ *(3 + nlength (3 ∗ H + idx ∗ H)) ^ 2*
by *simp*
then show *?thesis*
by *simp*
qed
also have ... $\leq$ *60 + 5553 ∗ H ^ 4 ∗ (3 + nlength (3 ∗ H + idx ∗ H))*^{2} +
10 ∗ H ^ 4 ∗ (3 + nlength (3 ∗ H + idx ∗ H)) + 4 ∗ nlength (Suc idx)
using *linear-le-pow* **by** *simp*
also have ... $\leq$ *60 + 5553 ∗ H ^ 4 ∗ (3 + nlength (3 ∗ H + idx ∗ H))*^{2} +
10 ∗ H ^ 4 ∗ (3 + nlength (3 ∗ H + idx ∗ H)) ^ 2 + 4 ∗ nlength (Suc idx)
using *linear-le-pow* **by** *simp*
also have ... = *60 + 5563 ∗ H ^ 4 ∗ (3 + nlength (3 ∗ H + idx ∗ H))*^{2} *+ 4 ∗ nlength (Suc idx)*
by *simp*
also have ... $\leq$ *60 + 5563 ∗ H ^ 4 ∗ (3 + nlength (3 ∗ H + idx ∗ H))*^{2} +
4 ∗ H^4 ∗ (3 + nlength (3 ∗ H + idx ∗ H))^2

proof −
have *idx* ≤ *idx* ∗ *H*
using *H* **by** *simp*
then have *Suc idx* ≤ *3* ∗ *H* + *idx* ∗ *H*
using *H* **by** *linarith*
then have *nlength* (*Suc idx*) ≤ *3* + *nlength* (*3* ∗ *H* + *idx* ∗ *H*)
using *nlength-mono trans-le-add2* **by** *presburger*
also have ... ≤ (*3* + *nlength* (*3* ∗ *H* + *idx* ∗ *H*)) ^ *2*
by (*simp add*: *power2-eq-square*)
also have ... ≤ *H* ∗ (*3* + *nlength* (*3* ∗ *H* + *idx* ∗ *H*)) ^ *2*
using *H* **by** *simp*
also have ... ≤ *H*^*4* ∗ (*3* + *nlength* (*3* ∗ *H* + *idx* ∗ *H*)) ^ *2*
using *linear-le-pow* **by** *simp*
finally have *nlength* (*Suc idx*) ≤ *H*^*4* ∗ (*3* + *nlength* (*3* ∗ *H* + *idx* ∗ *H*)) ^ *2* **.**
then show *?thesis*
by *simp*
qed
also have ... = *60* + *5567* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *idx* ∗ *H*))^2
by *simp*
also have ... ≤ *60* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *idx* ∗ *H*))^2 + *5567* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *idx* ∗ *H*))^2
using *H linear-le-pow* **by** *simp*
also have ... = *5627* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *idx* ∗ *H*))^2
by *simp*
finally have *?ttt* ≤ *ttt*
using *assms* **by** *simp*
then show *?thesis*
using *tm12 transforms-monotone* **by** *simp*
qed

end

end

lemma *transforms-tm-PHI0I*:
fixes *j* :: *tapeidx*
fixes *tps tps'* :: *tape list* **and** *ttt k idx H* :: *nat*
assumes *length tps* = *k* **and** *1* < *j* **and** *j* + *8* < *k* **and** *H* ≥ *3*
assumes
tps ! *1* = (⌊[]⌋, *1*)
tps ! *j* = (⌊*idx*⌋$_N$, *1*)
tps ! (*j* + *1*) = (⌊*H*⌋$_N$, *1*)
tps ! (*j* + *2*) = (⌊[]⌋, *1*)
tps ! (*j* + *3*) = (⌊[]⌋, *1*)
tps ! (*j* + *4*) = (⌊[]⌋, *1*)
tps ! (*j* + *5*) = (⌊[]⌋, *1*)
tps ! (*j* + *6*) = (⌊[]⌋, *1*)
tps ! (*j* + *7*) = (⌊[]⌋, *1*)
tps ! (*j* + *8*) = (⌊[]⌋, *1*)
assumes *tps'* = *tps*
[*j* := (⌊*Suc* (*Suc idx*)⌋$_N$, *1*),
j + *2* := (⌊*0*⌋$_N$, *1*),
j + *6* := (⌊*nll-Psi* (*Suc* (*Suc idx*) ∗ *H*) *H 0*⌋$_{NLL}$, *1*),
1 := *nlltape* (*nll-Psi* (*idx* ∗ *H*) *H 1* @ *nll-Psi* (*H* + *idx* ∗ *H*) *H 1* @ *nll-Psi* (*Suc* (*Suc idx*) ∗ *H*) *H 0*)]
assumes *ttt* = *5627* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *idx* ∗ *H*))^2
shows *transforms* (*tm-PHI0 j*) *tps ttt tps'*
proof −
interpret *loc*: *turing-machine-PHI0 j* **.**
show *?thesis*
using *loc.tps12-def loc.tm12' loc.tm12-eq-tm-PHI0 assms* **by** *metis*
qed

7.6.2 A Turing machine for Φ_1

The next TM expects a number H on tape $j + 1$ and appends to the formula on tape 1 the formula $\Psi([0, H), 1)$.

definition *tm-PHI1* :: *tapeidx* ⇒ *machine* **where**
  tm-PHI1 j ≡
    tm-setn (*j* + *2*) *1* ;;
    tm-Psigamma j ;;
    tm-extendl 1 (*j* + *6*)

lemma *tm-PHI1-tm*:
  assumes *0* < *j* **and** *j* + *7* < *k* **and** *G* ≥ *6*
  shows *turing-machine k G* (*tm-PHI1 j*)
  unfolding *tm-PHI1-def* **using** *assms tm-Psigamma-tm tm-setn-tm tm-extendl-tm* **by** *simp*

locale *turing-machine-PHI1* =
  fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-setn* (*j* + *2*) *1*
definition *tm2* ≡ *tm1* ;; *tm-Psigamma j*
definition *tm3* ≡ *tm2* ;; *tm-extendl 1* (*j* + *6*)

lemma *tm3-eq-tm-PHI1*: *tm3* = *tm-PHI1 j*
  using *tm3-def tm2-def tm1-def tm-PHI1-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *k idx H* :: *nat* **and** *nss* :: *nat list list*
  assumes *jk*: *length tps0* = *k 1* < *j j* + *7* < *k*
    and *H*: *H* ≥ *3*
  assumes *tps0*:
    tps0 ! *1* = *nlltape nss*
    tps0 ! *j* = (⌊*0*⌋$_N$, *1*)
    tps0 ! (*j* + *1*) = (⌊*H*⌋$_N$, *1*)
    tps0 ! (*j* + *2*) = (⌊[]⌋, *1*)
    tps0 ! (*j* + *3*) = (⌊[]⌋, *1*)
    tps0 ! (*j* + *4*) = (⌊[]⌋, *1*)
    tps0 ! (*j* + *5*) = (⌊[]⌋, *1*)
    tps0 ! (*j* + *6*) = (⌊[]⌋, *1*)
    tps0 ! (*j* + *7*) = (⌊[]⌋, *1*)
begin

definition *tps1* ≡ *tps0*
  [*j* + *2* := (⌊*1*⌋$_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
  assumes *ttt* = *12*
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def*
proof (*tform tps*: *tps0 tps1-def jk*)
  show *tps0* ! (*j* + *2*) = (⌊*0*⌋$_N$, *1*)
    using *tps0 jk canrepr-0* **by** *simp*
  show *ttt* = *10* + *2* ∗ *nlength 0* + *2* ∗ *nlength 1*
    using *assms canrepr-1* **by** *simp*
qed

definition *tps2* ≡ *tps0*
  [*j* + *2* := (⌊*1*⌋$_N$, *1*),
   j + *6* := (⌊*nll-Psi 0 H 1*⌋$_{NLL}$, *1*)]

lemma *tm2* [*transforms-intros*]:
  assumes *ttt* = *12* + *1851* ∗ *H* ^ *4*
  shows *transforms tm2 tps0 ttt tps2*

```
  unfolding tm2-def
proof (tform tps: tps0 H tps1-def tps2-def jk)
  show ttt = 12 + 1851 * H ^ 4 * (nlength (Suc 0))²
    using canrepr-1 assms by simp
qed

definition tps3 ≡ tps0
  [j + 2 := (⌊1⌋_N, 1),
   j + 6 := (⌊nll-Psi 0 H 1⌋_NLL, 1),
   1 := nlltape (nss @ nll-Psi 0 H 1)]

lemma tm3:
  assumes ttt = 16 + 1851 * H ^ 4 + 2 * nlllength (nll-Psi 0 H 1)
  shows transforms tm3 tps0 ttt tps3
  unfolding tm3-def by (tform tps: tps0 H tps2-def tps3-def jk time: assms)

lemma tm3':
  assumes ttt = 1875 * H ^ 4
  shows transforms tm3 tps0 ttt tps3
proof -
  let ?ttt = 16 + 1851 * H ^ 4 + 2 * nlllength (nll-Psi 0 H 1)
  have ?ttt ≤ 16 + 1851 * H ^ 4 + 2 * H * (3 + nlength H)
    using nlllength-nll-Psi-le
    by (metis (mono-tags, lifting) add-left-mono mult.assoc nat-mult-le-cancel1 plus-nat.add-0 rel-simps(51))
  also have ... = 16 + 1851 * H ^ 4 + 6 * H + 2 * H * nlength H
    by algebra
  also have ... ≤ 16 + 1851 * H ^ 4 + 6 * H + 2 * H * H
    using nlength-le-n by simp
  also have ... ≤ 16 + 1851 * H ^ 4 + 6 * H * H + 2 * H * H
    by simp
  also have ... = 16 + 1851 * H ^ 4 + 8 * H^2
    by algebra
  also have ... ≤ 16 + 1851 * H ^ 4 + 8 * H^4
    using pow-mono'[of 2 4 H] by simp
  also have ... = 16 + 1859 * H ^ 4
    by simp
  also have ... ≤ 16 * H^4 + 1859 * H ^ 4
    using H by simp
  also have ... = 1875 * H ^ 4
    by simp
  finally have ?ttt ≤ 1875 * H ^ 4 .
  then show ?thesis
    using assms tm3 transforms-monotone by simp
qed

end

end

lemma transforms-tm-PHI1I:
  fixes j :: tapeidx
  fixes tps tps' :: tape list and ttt k H :: nat and nss :: nat list list
  assumes length tps = k and 1 < j and j + 7 < k and H ≥ 3
  assumes
    tps ! 1 = nlltape nss
    tps ! j = (⌊0⌋_N, 1)
    tps ! (j + 1) = (⌊H⌋_N, 1)
    tps ! (j + 2) = (⌊[]⌋, 1)
    tps ! (j + 3) = (⌊[]⌋, 1)
    tps ! (j + 4) = (⌊[]⌋, 1)
    tps ! (j + 5) = (⌊[]⌋, 1)
    tps ! (j + 6) = (⌊[]⌋, 1)
    tps ! (j + 7) = (⌊[]⌋, 1)
```

assumes *tps'* = *tps*
  [*j* + *2* := (⌊*1*⌋$_N$, *1*),
   j + *6* := (⌊*nll-Psi 0 H 1*⌋$_{NLL}$, *1*),
   1 := *nlltape* (*nss* @ *nll-Psi 0 H 1*)]
assumes *ttt* = *1875* ∗ *H* ^ *4*
shows *transforms* (*tm-PHI1 j*) *tps ttt tps'*
proof −
  interpret *loc*: *turing-machine-PHI1 j* **.**
  show *?thesis*
    using *loc.tps3-def loc.tm3' loc.tm3-eq-tm-PHI1 assms* **by** *metis*
qed

7.6.3 A Turing machine for Φ_2

The next TM expects a number i on tape j and a number H on tape $j+1$. It appends to the formula on tape 1 the formula $\Psi([(2i+1)H,(2i+2)H),3) \wedge \Psi([(2i+2)H,(2i+3)H),3)$.

definition *tm-PHI2* :: *tapeidx* ⇒ *machine* **where**
  tm-PHI2 j ≡
    tm-times2 j ;;
    tm-incr j ;;
    tm-setn (*j* + *2*) *3* ;;
    tm-Psigamma j ;;
    tm-extendl-erase 1 (*j* + *6*) ;;
    tm-incr j ;;
    tm-Psigamma j ;;
    tm-extendl 1 (*j* + *6*)

lemma *tm-PHI2-tm*:
  assumes *0* < *j* **and** *j* + *8* < *k* **and** *G* ≥ *6*
  shows *turing-machine k G* (*tm-PHI2 j*)
  unfolding *tm-PHI2-def*
  using *assms tm-Psigamma-tm tm-extendl-tm tm-erase-cr-tm tm-times2-tm tm-incr-tm tm-setn-tm tm-cr-tm tm-extendl-erase-tm*
  by *simp*

locale *turing-machine-PHI2* =
  fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-times2 j*
definition *tm2* ≡ *tm1* ;; *tm-incr j*
definition *tm3* ≡ *tm2* ;; *tm-setn* (*j* + *2*) *3*
definition *tm4* ≡ *tm3* ;; *tm-Psigamma j*
definition *tm5* ≡ *tm4* ;; *tm-extendl-erase 1* (*j* + *6*)
definition *tm7* ≡ *tm5* ;; *tm-incr j*
definition *tm8* ≡ *tm7* ;; *tm-Psigamma j*
definition *tm9* ≡ *tm8* ;; *tm-extendl 1* (*j* + *6*)

lemma *tm9-eq-tm-PHI2*: *tm9* = *tm-PHI2 j*
  using *tm9-def tm8-def tm7-def tm5-def tm4-def tm3-def tm2-def tm1-def tm-PHI2-def*
  by *simp*

context
  fixes *tps0* :: *tape list* **and** *k idx H* :: *nat* **and** *nss* :: *nat list list*
  assumes *jk*: *length tps0* = *k 1* < *j j* + *7* < *k*
    and *H*: *H* ≥ *3*
  assumes *tps0*:
    tps0 ! *1* = *nlltape nss*
    tps0 ! *j* = (⌊*idx*⌋$_N$, *1*)
    tps0 ! (*j* + *1*) = (⌊*H*⌋$_N$, *1*)
    tps0 ! (*j* + *2*) = (⌊[]⌋, *1*)
    tps0 ! (*j* + *3*) = (⌊[]⌋, *1*)
    tps0 ! (*j* + *4*) = (⌊[]⌋, *1*)

```
    tps0 ! (j + 5) = (⌊[]⌋, 1)
    tps0 ! (j + 6) = (⌊[]⌋, 1)
    tps0 ! (j + 7) = (⌊[]⌋, 1)
begin

definition tps1 ≡ tps0
  [j := (⌊2 * idx⌋_N, 1)]

lemma tm1 [transforms-intros]:
  assumes ttt = 5 + 2 * nlength idx
  shows transforms tm1 tps0 ttt tps1
  unfolding tm1-def by (tform tps: tps0 tps1-def jk assms)

definition tps2 ≡ tps0
  [j := (⌊2 * idx + 1⌋_N, 1)]

lemma tm2:
  assumes ttt = 10 + 2 * nlength idx + 2 * nlength (2 * idx)
  shows transforms tm2 tps0 ttt tps2
  unfolding tm2-def by (tform tps: tps0 H tps1-def tps2-def jk assms)

lemma tm2' [transforms-intros]:
  assumes ttt = 12 + 4 * nlength idx
  shows transforms tm2 tps0 ttt tps2
proof -
  have 10 + 2 * nlength idx + 2 * nlength (2 * idx) ≤ 10 + 2 * nlength idx + 2 * (Suc (nlength idx))
    using nlength-times2 by (meson add-left-mono mult-le-mono2)
  then have 10 + 2 * nlength idx + 2 * nlength (2 * idx) ≤ ttt
    using assms by simp
  then show ?thesis
    using tm2 transforms-monotone by simp
qed

definition tps3 ≡ tps0
  [j := (⌊2 * idx + 1⌋_N, 1),
   j + 2 := (⌊3⌋_N, 1)]

lemma tm3 [transforms-intros]:
  assumes ttt = 26 + 4 * nlength idx
  shows transforms tm3 tps0 ttt tps3
  unfolding tm3-def
proof (tform tps: tps0 H tps2-def tps3-def jk)
  show tps2 ! (j + 2) = (⌊0⌋_N, 1)
    using tps2-def jk canrepr-0 tps0 by simp
  show ttt = 12 + 4 * nlength idx + (10 + 2 * nlength 0 + 2 * nlength 3)
    using nlength-3 assms by simp
qed

definition tps4 ≡ tps0
  [j := (⌊2 * idx + 1⌋_N, 1),
   j + 2 := (⌊3⌋_N, 1),
   j + 6 := (⌊nll-Psi (Suc (2 * idx) * H) H 3⌋_NLL, 1)]

lemma tm4 [transforms-intros]:
  assumes ttt = 26 + 4 * nlength idx + 1851 * H ^ 4 * (nlength (Suc (Suc (2 * idx))))^2
  shows transforms tm4 tps0 ttt tps4
  unfolding tm4-def by (tform tps: tps0 H tps3-def tps4-def jk time: assms)

definition tps5 ≡ tps0
  [j := (⌊2 * idx + 1⌋_N, 1),
   j + 2 := (⌊3⌋_N, 1),
   j + 6 := (⌊[]⌋, 1),
   1 := nlltape (nss @ nll-Psi (H + 2 * idx * H) H 3)]
```

lemma *tm5* [*transforms-intros*]:
  assumes $ttt = 37 + 4 * nlength\ idx + 1851 * H \,\hat{}\, 4 * (nlength\ (Suc\ (Suc\ (2 * idx))))^2 +$
    $4 * nlllength\ (nll\text{-}Psi\ (H + 2 * idx * H)\ H\ 3)$
  shows *transforms tm5 tps0 ttt tps5*
  unfolding *tm5-def* **by** (*tform tps*: *tps0 H tps4-def tps5-def jk time*: *assms*)

definition *tps7* ≡ *tps0*
  $[j := (\lfloor 2 * idx + 2 \rfloor_N,\ 1),$
   $j + 2 := (\lfloor 3 \rfloor_N,\ 1),$
   $j + 6 := (\lfloor [] \rfloor,\ 1),$
   $1 := nlltape\ (nss\ @\ nll\text{-}Psi\ (H + 2 * idx * H)\ H\ 3)]$

lemma *tm7*:
  assumes $ttt = 42 + 4 * nlength\ idx + 1851 * H \,\hat{}\, 4 * (nlength\ (Suc\ (Suc\ (2 * idx))))^2 +$
   $4 * nlllength\ (nll\text{-}Psi\ (H + 2 * idx * H)\ H\ 3) + 2 * nlength\ (Suc\ (2 * idx))$
  shows *transforms tm7 tps0 ttt tps7*
  unfolding *tm7-def*
proof (*tform tps*: *tps0 H tps5-def tps7-def jk time*: *assms*)
  show $tps7 = tps5[j := (\lfloor Suc\ (Suc\ (2 * idx)) \rfloor_N,\ 1)]$
    using *tps5-def tps7-def jk* **by** (*simp add*: *list-update-swap*)
qed

lemma *tm7'* [*transforms-intros*]:
  assumes $ttt = 44 + 6 * nlength\ idx + 1851 * H \,\hat{}\, 4 * (nlength\ (Suc\ (Suc\ (2 * idx))))^2 +$
    $4 * nlllength\ (nll\text{-}Psi\ (H + 2 * idx * H)\ H\ 3)$
  shows *transforms tm7 tps0 ttt tps7*
proof −
  let $?ttt = 42 + 4 * nlength\ idx + 1851 * H \,\hat{}\, 4 * (nlength\ (Suc\ (Suc\ (2 * idx))))^2 +$
    $4 * nlllength\ (nll\text{-}Psi\ (H + 2 * idx * H)\ H\ 3) +$
    $2 * nlength\ (Suc\ (2 * idx))$
  have $?ttt \le 42 + 4 * nlength\ idx + 1851 * H \,\hat{}\, 4 * (nlength\ (Suc\ (Suc\ (2 * idx))))^2 +$
    $4 * nlllength\ (nll\text{-}Psi\ (H + 2 * idx * H)\ H\ 3) +$
    $2 * (Suc\ (nlength\ idx))$
   using *nlength-times2plus1* **by** (*metis add.commute add-left-mono mult-le-mono2 plus-1-eq-Suc*)
  then have $?ttt \le ttt$
   using *assms* **by** *simp*
  then show *?thesis*
   using *tm7 transforms-monotone* **by** *simp*
qed

definition *tps8* ≡ *tps0*
  $[j := (\lfloor 2 * idx + 2 \rfloor_N,\ 1),$
   $j + 2 := (\lfloor 3 \rfloor_N,\ 1),$
   $j + 6 := (\lfloor nll\text{-}Psi\ (Suc\ (Suc\ (2 * idx)) * H)\ H\ 3 \rfloor_{NLL},\ 1),$
   $1 := nlltape\ (nss\ @\ nll\text{-}Psi\ (H + 2 * idx * H)\ H\ 3)]$

lemma *tm8* [*transforms-intros*]:
  assumes $ttt = 44 + 6 * nlength\ idx + 1851 * H \,\hat{}\, 4 * (nlength\ (Suc\ (Suc\ (2 * idx))))^2 +$
    $4 * nlllength\ (nll\text{-}Psi\ (H + 2 * idx * H)\ H\ 3) +$
    $1851 * H \,\hat{}\, 4 * (nlength\ (Suc\ (Suc\ (Suc\ (2 * idx)))))^2$
  shows *transforms tm8 tps0 ttt tps8*
  unfolding *tm8-def*
proof (*tform tps*: *tps0 H tps7-def tps8-def jk time*: *assms*)
  show *tps8* = *tps7*
    $[j + 6 := (\lfloor nll\text{-}Psi\ (Suc\ (Suc\ (2 * idx)) * H)\ H\ 3 \rfloor_{NLL},\ 1)]$
   unfolding *tps8-def tps7-def* **by** (*simp add*: *list-update-swap*[*of j+6*])
qed

definition *tps9* ≡ *tps0*
  $[j := (\lfloor 2 * idx + 2 \rfloor_N,\ 1),$
   $j + 2 := (\lfloor 3 \rfloor_N,\ 1),$
   $j + 6 := (\lfloor nll\text{-}Psi\ (Suc\ (Suc\ (2 * idx)) * H)\ H\ 3 \rfloor_{NLL},\ 1),$

1 := nlltape (nss @ nll-Psi (H + 2 * idx * H) H 3 @ nll-Psi (2 * H + 2 * idx * H) H 3)]

lemma tm9:
assumes ttt = 48 + 6 * nlength idx + 1851 * H ^ 4 * (nlength (Suc (Suc (2 * idx))))^2 +
4 * nlllength (nll-Psi (H + 2 * idx * H) H 3) +
1851 * H ^ 4 * (nlength (Suc (Suc (Suc (2 * idx)))))^2 +
2 * nlllength (nll-Psi (2 * H + 2 * idx * H) H 3)
shows transforms tm9 tps0 ttt tps9
unfolding tm9-def **by** (tform tps: tps0 H tps9-def tps8-def jk time: assms)

lemma tm9′:
assumes ttt = 3764 * H ^ 4 * (3 + nlength (3 * H + 2 * idx * H))^2
shows transforms tm9 tps0 ttt tps9
proof −
let ?ttt = 48 + 6 * nlength idx + 1851 * H ^ 4 * (nlength (Suc (Suc (2 * idx))))^2 +
4 * nlllength (nll-Psi (H + 2 * idx * H) H 3) + 1851 * H ^ 4 * (nlength (Suc (Suc (Suc (2 * idx)))))^2 +
2 * nlllength (nll-Psi (2 * H + 2 * idx * H) H 3)
have ?ttt ≤ 48 + 6 * nlength idx + 1851 * H ^ 4 * (nlength (Suc (Suc (2 * idx))))^2 +
4 * H * (3 + nlength (2 * H + 2 * idx * H + H)) + 1851 * H ^ 4 * (nlength (Suc (Suc (Suc (2 * idx)))))^2 +
2 * nlllength (nll-Psi (2 * H + 2 * idx * H) H 3)
using nlllength-nll-Psi-le′[of H + 2 * idx * H 2 * H + 2 * idx * H H 3] **by** simp
also have ... ≤ 48 + 6 * nlength idx + 1851 * H ^ 4 * (nlength (Suc (Suc (2 * idx))))^2 +
5 * H * (3 + nlength (2 * H + 2 * idx * H + H)) + 1851 * H ^ 4 * (nlength (Suc (Suc (Suc (2 * idx)))))^2 +
3 * H * (3 + nlength (2 * H + 2 * idx * H + H))
using nlllength-nll-Psi-le′[of 2 * H + 2 * idx * H 2 * H + 2 * idx * H H 3] **by** simp
also have ... = 48 + 6 * nlength idx +
1851 * H ^ 4 * (nlength (Suc (Suc (2 * idx))))^2 +
8 * H * (3 + nlength (2 * H + 2 * idx * H + H)) +
1851 * H ^ 4 * (nlength (Suc (Suc (Suc (2 * idx)))))^2
by simp
also have ... ≤ 48 + 6 * nlength idx +
1851 * H ^ 4 * (nlength (Suc (Suc (Suc (2 * idx)))))^2 +
8 * H * (3 + nlength (2 * H + 2 * idx * H + H)) +
1851 * H ^ 4 * (nlength (Suc (Suc (Suc (2 * idx)))))^2
using H4-nlength H **by** simp
also have ... = 48 + 6 * nlength idx + 3702 * H ^ 4 * (nlength (Suc (Suc (Suc (2 * idx)))))^2 +
8 * H * (3 + nlength (3 * H + 2 * idx * H))
by simp
also have ... ≤ 48 + 6 * nlength idx + 3702 * H ^ 4 * (3 + nlength (3 * H + 2 * idx * H))^2 +
8 * H * (3 + nlength (3 * H + 2 * idx * H))
proof −
have Suc (Suc (Suc (2 * idx))) ≤ 3 * H + 2 * idx * H
using H
by (metis One-nat-def Suc3-eq-add-3 Suc-eq-plus1-left add-leD1 comm-monoid-mult-class.mult-1 distrib-right mult.commute mult-le-mono1)
then have nlength (Suc (Suc (Suc (2 * idx)))) ≤ 3 + nlength (3 * H + 2 * idx * H)
using nlength-mono trans-le-add2 **by** blast
then show ?thesis
by simp
qed
also have ... ≤ 48 + 6 * nlength idx + 3702 * H ^ 4 * (3 + nlength (3 * H + 2 * idx * H))^2 +
8 * H^4 * (3 + nlength (3 * H + 2 * idx * H))
using linear-le-pow **by** simp
also have ... ≤ 48 + 6 * nlength idx + 3702 * H ^ 4 * (3 + nlength (3 * H + 2 * idx * H))^2 +
8 * H^4 * (3 + nlength (3 * H + 2 * idx * H))^2
using linear-le-pow **by** simp
also have ... = 48 + 6 * nlength idx + 3710 * H ^ 4 * (3 + nlength (3 * H + 2 * idx * H))^2
by simp
also have ... ≤ 48 + 3716 * H ^ 4 * (3 + nlength (3 * H + 2 * idx * H))^2
proof −
have nlength idx ≤ nlength (3 * H + 2 * idx * H)

```
      using H by (intro nlength-mono) (simp add: trans-le-add2)
    also have ... ≤ 3 + nlength (3 ∗ H + 2 ∗ idx ∗ H)
      by simp
    also have ... ≤ (3 + nlength (3 ∗ H + 2 ∗ idx ∗ H)) ^ 2
      using linear-le-pow by simp
    also have ... ≤ H ^ 4 ∗ (3 + nlength (3 ∗ H + 2 ∗ idx ∗ H)) ^ 2
      using H by simp
    finally have nlength idx ≤ H ^ 4 ∗ (3 + nlength (3 ∗ H + 2 ∗ idx ∗ H)) ^ 2 .
    then show ?thesis
      by simp
  qed
  also have ... ≤ 3764 ∗ H ^ 4 ∗ (3 + nlength (3 ∗ H + 2 ∗ idx ∗ H))²
  proof −
    have 1 ≤ nlength (3 ∗ H + 2 ∗ idx ∗ H)
      using H nlength-0
      by (metis One-nat-def Suc-leI add-eq-0-iff-both-eq-0 length-0-conv length-greater-0-conv mult-Suc
        not-numeral-le-zero numeral-3-eq-3)
    also have ... ≤ 3 + nlength (3 ∗ H + 2 ∗ idx ∗ H)
      by simp
    also have ... ≤ (3 + nlength (3 ∗ H + 2 ∗ idx ∗ H)) ^ 2
      using linear-le-pow by simp
    also have ... ≤ H ^ 4 ∗ (3 + nlength (3 ∗ H + 2 ∗ idx ∗ H)) ^ 2
      using H by simp
    finally have 1 ≤ H ^ 4 ∗ (3 + nlength (3 ∗ H + 2 ∗ idx ∗ H)) ^ 2 .
    then show ?thesis
      by simp
  qed
  finally have ?ttt ≤ 3764 ∗ H ^ 4 ∗ (3 + nlength (3 ∗ H + 2 ∗ idx ∗ H))² .
  then show ?thesis
    using assms tm9 transforms-monotone by simp
qed

end

end

lemma transforms-tm-PHI2I:
  fixes j :: tapeidx
  fixes tps tps' :: tape list and ttt k idx H :: nat and nss :: nat list list
  assumes length tps = k and 1 < j and j + 8 < k
    and H ≥ 3
  assumes
    tps ! 1 = nlltape nss
    tps ! j = (⌊idx⌋N, 1)
    tps ! (j + 1) = (⌊H⌋N, 1)
    tps ! (j + 2) = (⌊[]⌋, 1)
    tps ! (j + 3) = (⌊[]⌋, 1)
    tps ! (j + 4) = (⌊[]⌋, 1)
    tps ! (j + 5) = (⌊[]⌋, 1)
    tps ! (j + 6) = (⌊[]⌋, 1)
    tps ! (j + 7) = (⌊[]⌋, 1)
    tps ! (j + 8) = (⌊[]⌋, 1)
  assumes ttt = 3764 ∗ H ^ 4 ∗ (3 + nlength (3 ∗ H + 2 ∗ idx ∗ H))²
  assumes tps' = tps
    [j := (⌊2 ∗ idx + 2⌋N, 1),
     j + 2 := (⌊3⌋N, 1),
     j + 6 := (⌊nll-Psi (Suc (Suc (2 ∗ idx)) ∗ H) H 3⌋NLL, 1),
     1 := nlltape (nss @ nll-Psi (H + 2 ∗ idx ∗ H) H 3 @ nll-Psi (2 ∗ H + 2 ∗ idx ∗ H) H 3)]
  shows transforms (tm-PHI2 j) tps ttt tps'
proof −
  interpret loc: turing-machine-PHI2 j .
  show ?thesis
    using loc.tm9' loc.tps9-def loc.tm9-eq-tm-PHI2 assms by simp
```

qed

7.6.4 Turing machines for Φ_3, Φ_4, and Φ_5

The CNF formulas Φ_3, Φ_4, and Φ_5 have a similar structure and can thus be handled by the same Turing machine. The following TM has a parameter *step* and the usual tape parameter j. It expects on tape j a number *idx*, on tape $j + 1$ a number H, on tape $j + 2$ a number κ, and on tape $j + 8$ the number $idx + step \cdot numiter$ for some number *numiter*. It appends to the CNF formula on tape 1 the formula $\Psi(\gamma_{idx}, \kappa) \wedge \Psi(\gamma_{idx+step\cdot(numiter-1)}, \kappa)$, where $\gamma_i = [iH, (i + 1)H)$.

definition *tm-PHI345* :: *nat* ⇒ *tapeidx* ⇒ *machine* **where**
  tm-PHI345 step j ≡
    WHILE tm-equalsn j (j + 8) (j + 3) ; λrs. rs ! (j + 3) = □ DO
      tm-Psigamma j ;;
      tm-extendl-erase 1 (j + 6) ;;
      tm-plus-const step j
    DONE

lemma *tm-PHI345-tm*:
  assumes *G ≥ 6* **and** *0 < j* **and** *j + 8 < k*
  shows *turing-machine k G (tm-PHI345 step j)*
  unfolding *tm-PHI345-def*
  using *assms tm-equalsn-tm tm-Psigamma-tm tm-extendl-erase-tm tm-plus-const-tm*
    turing-machine-loop-turing-machine
  by *simp*

locale *turing-machine-PHI345* =
  fixes *step* :: *nat* **and** *j* :: *tapeidx*
begin

definition *tmC* ≡ *tm-equalsn j (j + 8) (j + 3)*
definition *tm1* ≡ *tm-Psigamma j*
definition *tm2* ≡ *tm1 ;; tm-extendl-erase 1 (j + 6)*
definition *tm4* ≡ *tm2 ;; tm-plus-const step j*
definition *tmL* ≡ *WHILE tmC ; λrs. rs ! (j + 3) = □ DO tm4 DONE*

lemma *tmL-eq-tm-PHI345*: *tmL = tm-PHI345 step j*
  unfolding *tmL-def tm4-def tm2-def tm1-def tm-PHI345-def tmC-def* **by** *simp*

context
  fixes *tps0* :: *tape list* **and** *numiter H k idx kappa* :: *nat* **and** *nss* :: *nat list list*
  assumes *jk*: *length tps0 = k 1 < j j + 8 < k*
    and *H*: *H ≥ 3*
    and *kappa*: *kappa ≤ H*
    and *step*: *step > 0*
  assumes *tps0*:
    tps0 ! 1 = nlltape nss
    tps0 ! j = (⌊idx⌋_N, 1)
    tps0 ! (j + 1) = (⌊H⌋_N, 1)
    tps0 ! (j + 2) = (⌊kappa⌋_N, 1)
    tps0 ! (j + 3) = (⌊[]⌋, 1)
    tps0 ! (j + 4) = (⌊[]⌋, 1)
    tps0 ! (j + 5) = (⌊[]⌋, 1)
    tps0 ! (j + 6) = (⌊[]⌋, 1)
    tps0 ! (j + 7) = (⌊[]⌋, 1)
    *tps0 ! (j + 8) = (⌊idx + step * numiter⌋_N, 1)*
begin

definition *tpsL* :: *nat* ⇒ *tape list* **where**
  tpsL t ≡ *tps0*
    *[j := (⌊idx + step * t⌋_N, 1),*
     *1 := nlltape (nss @ concat (map (λi. nll-Psi (H * (idx + step * i)) H kappa) [0..<t]))]*

lemma *tpsL-eq-tps0*: *tpsL 0 = tps0*
proof −
  have $\lfloor idx \rfloor_N = \lfloor idx + step * 0 \rfloor_N$
    by *simp*
  moreover have *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*0*])) = *nlltape nss*
    using *nllcontents-Nil* **by** *simp*
  ultimately show *?thesis*
    using *tpsL-def tps0 jk* **by** (*metis list-update-id*)
qed

definition *tpsC* :: *nat* ⇒ *tape list* **where**
  tpsC t ≡ *tps0*
    [*j* := ($\lfloor idx + step * t \rfloor_N$, *1*),
     1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*t*])),
     j + *3* := ($\lfloor t = numiter \rfloor_B$, *1*)]

lemma *tmC*:
  assumes *t* ≤ *numiter*
    and *ttt* = *3* ∗ *nlength* (*idx* + *step* ∗ *t*) + *7*
  shows *transforms tmC* (*tpsL t*) *ttt* (*tpsC t*)
  unfolding *tmC-def*
proof (*tform tps*: *tps0 tpsL-def jk*)
  show *tpsL t* ! (*j* + *3*) = ($\lfloor 0 \rfloor_N$, *1*)
    using *canrepr-0 jk tpsL-def tps0* **by** *simp*
  show (*0*::*nat*) ≤ *1*
    by *simp*
  show *tpsC t* = (*tpsL t*)
      [*j* + *3* := ($\lfloor idx + step * t = idx + step * numiter \rfloor_B$, *1*)]
    using *step tpsC-def jk tpsL-def tps0* **by** *simp*
  show *ttt* = *3* ∗ *nlength* (*min* (*idx* + *step* ∗ *t*) (*idx* + *step* ∗ *numiter*)) + *7*
    using *assms* **by** (*simp add*: *min-def*)
qed

lemma *tmC′* [*transforms-intros*]:
  assumes *t* ≤ *numiter*
    and *ttt* = *3* ∗ *nlength* (*idx* + *step* ∗ *numiter*) + *7*
  shows *transforms tmC* (*tpsL t*) *ttt* (*tpsC t*)
proof −
  have *3* ∗ *nlength* (*idx* + *step* ∗ *t*) + *7* ≤ *ttt*
    using *assms nlength-mono* **by** *simp*
  then show *?thesis*
    using *assms tmC transforms-monotone* **by** *blast*
qed

definition *tpsL0* :: *nat* ⇒ *tape list* **where**
  tpsL0 t ≡ *tps0*
    [*j* := ($\lfloor idx + step * t \rfloor_N$, *1*),
     1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*t*]))]

lemma *tpsL0-eq-tpsC*:
  assumes *t* < *numiter*
  shows *tpsL0 t* = *tpsC t*
  unfolding *tpsL0-def tpsC-def*
  using *assms jk ncontents-0 tps0 list-update-id*[*of tps0 j* + *3*]
  by (*simp add*: *list-update-swap*)

definition *tpsL1* :: *nat* ⇒ *tape list* **where**
  tpsL1 t ≡ *tps0*
    [*j* := ($\lfloor idx + step * t \rfloor_N$, *1*),
     1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*t*])),
     j + *6* := ($\lfloor nll\text{-}Psi\ ((idx{+}step{*}t) * H)\ H\ kappa \rfloor_{NLL}$, *1*)]

lemma *tm1* [*transforms-intros*]:
assumes $t < numiter$
and $ttt = 1851 * H \,\hat{}\, 4 * (nlength (Suc (idx+step*t)))^2$
shows *transforms tm1* (*tpsL0 t*) *ttt* (*tpsL1 t*)
unfolding *tm1-def* **by** (*tform tps*: *H kappa tps0 tpsL0-def tpsL1-def jk time*: *assms*(*2*))

definition *tpsL2* :: *nat* ⇒ *tape list* **where**
tpsL2 t ≡ *tps0*
[$j := (\lfloor idx + step * t\rfloor_N, 1)$,
1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*t*])),
$j + 6 := (\lfloor[]\rfloor, 1)$,
1 := *nlltape* ((*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*t*])) @ *nll-Psi* ((*idx*+*step*∗*t*) ∗ *H*) *H kappa*)]

lemma *tm2*:
assumes $t < numiter$
and $ttt = 1851 * H \,\hat{}\, 4 * (nlength (Suc (idx + step * t)))^2 +$
$(11 + 4 * nlllength (nll\text{-}Psi ((idx + step * t) * H) H kappa))$
shows *transforms tm2* (*tpsL0 t*) *ttt* (*tpsL2 t*)
unfolding *tm2-def* **by** (*tform tps*: *assms H kappa tps0 tpsL1-def tpsL2-def jk*)

definition *tpsL2′* :: *nat* ⇒ *tape list* **where**
tpsL2′ t ≡ *tps0*
[$j := (\lfloor idx + step * t\rfloor_N, 1)$,
$j + 6 := (\lfloor[]\rfloor, 1)$,
1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*t*]) @ *nll-Psi* ((*idx*+*step*∗*t*) ∗ *H*) *H kappa*)]

lemma *tpsL2′*: *tpsL2 t* = *tpsL2′ t*
unfolding *tpsL2-def tpsL2′-def* **by** (*simp only*: *list-update-swap list-update-overwrite*) *simp*

lemma *tm2′*:
assumes $t < numiter$
and $ttt = 1851 * H \,\hat{}\, 4 * (nlength (idx + step * numiter))^2 +$
$(11 + 4 * nlllength (nll\text{-}Psi ((idx + step * t) * H) H kappa))$
shows *transforms tm2* (*tpsL0 t*) *ttt* (*tpsL2′ t*)
proof −
let $?ttt = 1851 * H \,\hat{}\, 4 * (nlength (Suc (idx + step * t)))^2 +$
$(11 + 4 * nlllength (nll\text{-}Psi ((idx + step * t) * H) H kappa))$
have $?ttt \le 1851 * H \,\hat{}\, 4 * (nlength (idx + step * numiter))^2 +$
$(11 + 4 * nlllength (nll\text{-}Psi ((idx + step * t) * H) H kappa))$
proof −
have $Suc (idx + step * t) \le Suc (idx + step * (numiter - 1))$
using *assms*(*1*) *step* **by** *simp*
also have ... $= Suc (idx + step * numiter - step)$
by (*metis Nat.add-diff-assoc One-nat-def Suc-le-eq add-less-same-cancel1 assms*(*1*) *mult.right-neutral nat-mult-le-cancel-disj nat-neq-iff not-add-less1 right-diff-distrib′*)
also have ... $\le idx + step * numiter$
using *step Suc-le-eq assms*(*1*) **by** *simp*
finally have $Suc (idx + step * t) \le idx + step * numiter$ **.**
then have $nlength (Suc (idx + step * t)) \le nlength (idx + step * numiter)$
using *nlength-mono* **by** *simp*
then show *?thesis*
by *simp*
qed
then have *transforms tm2* (*tpsL0 t*) *ttt* (*tpsL2 t*)
using *assms tm2 transforms-monotone* **by** *blast*
then show *?thesis*
using *tpsL2′* **by** *simp*
qed

definition *tpsL2′′* :: *nat* ⇒ *tape list* **where**

tpsL2'' t ≡ *tps0*
  [*j* := (⌊*idx* + *step* ∗ *t*⌋$_N$, *1*),
   1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*Suc t*])),
   j + *6* := (⌊[]⌋, *1*)]

lemma *tpsL2''*: *tpsL2'' t* = *tpsL2' t*
proof −
  have *nll-Psi* ((*idx*+*step*∗*t*) ∗ *H*) *H kappa* = *nll-Psi* (*H* ∗ (*idx*+*step*∗*t*)) *H kappa*
    by (*simp add*: *mult.commute*)
  then have *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*t*]) @ *nll-Psi* ((*idx*+*step*∗*t*) ∗ *H*) *H kappa* =
      concat (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*Suc t*])
    by *simp*
  then show *tpsL2'' t* = *tpsL2' t*
    using *tpsL2'-def tpsL2''-def* **by** (*simp add*: *list-update-swap*)
qed

lemma *nlllength-nll-Psi*:
  assumes *t* < *numiter*
  shows *nlllength* (*nll-Psi* ((*idx* + *step* ∗ *t*) ∗ *H*) *H kappa*) ≤ *5* ∗ *H* ^ *4* ∗ *nlength* (*idx* + *step* ∗ *numiter*)^*2*
proof −
  have *nlllength* (*nll-Psi* ((*idx* + *step* ∗ *t*) ∗ *H*) *H kappa*) ≤ *H* ∗ (*3* + *nlength* ((*idx* + *step* ∗ *t*) ∗ *H* + *H*))
    using *nlllength-nll-Psi-le* **by** *simp*
  also have ... ≤ *H* ∗ (*3* + *nlength* ((*idx* + *step* ∗ *numiter*) ∗ *H*))
  proof −
    have (*idx* + *1* + *step* ∗ *t*) ≤ (*idx* + *step* ∗ *Suc t*)
      using *step* **by** *simp*
    then have (*idx* + *1* + *step* ∗ *t*) ≤ (*idx* + *step* ∗ *numiter*)
      using *assms*(*1*) *Suc-le-eq* **by** *auto*
    then have (*idx* + *1* + *step* ∗ *t*) ∗ *H* ≤ (*idx* + *step* ∗ *numiter*) ∗ *H*
      using *mult-le-cancel2* **by** *blast*
    then show *?thesis*
      using *nlength-mono* **by** *simp*
  qed
  also have ... = *3* ∗ *H* + *H* ∗ *nlength* ((*idx* + *step* ∗ *numiter*) ∗ *H*)
    by *algebra*
  also have ... ≤ *3* ∗ *H* + *H* ∗ (*nlength* (*idx* + *step* ∗ *numiter*) + *nlength H*)
    using *nlength-prod* **by** *simp*
  also have ... ≤ *3* ∗ *H* + *H* ∗ (*nlength* (*idx* + *step* ∗ *numiter*) + *H*)
    using *nlength-le-n* **by** *simp*
  also have ... = *3* ∗ *H* + *H* ^ *2* + *H* ∗ *nlength* (*idx* + *step* ∗ *numiter*)
    by *algebra*
  also have ... ≤ *3* ∗ *H*^*4* + *H* ^ *2* + *H* ∗ *nlength* (*idx* + *step* ∗ *numiter*)
    using *linear-le-pow* **by** *simp*
  also have ... ≤ *3* ∗ *H*^*4* + *H* ^ *4* + *H* ∗ *nlength* (*idx* + *step* ∗ *numiter*)
    using *pow-mono'* **by** *simp*
  also have ... ≤ *4* ∗ *H*^*4* + *H* ^ *4* ∗ *nlength* (*idx* + *step* ∗ *numiter*)
    using *linear-le-pow* **by** *simp*
  also have ... ≤ *4* ∗ *H*^*4* + *H* ^ *4* ∗ *nlength* (*idx* + *step* ∗ *numiter*)^*2*
    using *linear-le-pow* **by** *simp*
  also have ... ≤ *5* ∗ *H* ^ *4* ∗ *nlength* (*idx* + *step* ∗ *numiter*)^*2*
  proof −
    have *idx* + *step* ∗ *numiter* > *0*
      using *assms*(*1*) *step* **by** *simp*
    then have *nlength* (*idx* + *step* ∗ *numiter*) > *0*
      using *nlength-0* **by** *simp*
    then have *nlength* (*idx* + *step* ∗ *numiter*) ^ *2* > *0*
      by *simp*
    then have *H* ^ *4* ≤ *H* ^ *4* ∗ *nlength* (*idx* + *step* ∗ *numiter*) ^ *2*
      by (*metis One-nat-def Suc-leI mult-numeral-1-right nat-mult-le-cancel-disj numerals*(*1*))
    then show *?thesis*
      by *simp*
  qed

finally show *?thesis* **.**
qed

lemma *tm2''* [*transforms-intros*]:
assumes $t < numiter$ **and** $ttt = 1871 * H \hat{}\ 4 * (nlength\ (idx + step * numiter))^2 + 11$
shows *transforms tm2 (tpsL0 t) ttt (tpsL2'' t)*
proof −
have *transforms tm2 (tpsL0 t) ttt (tpsL2' t)*
using *tm2'[OF assms(1)] nlllength-nll-Psi[OF assms(1)] transforms-monotone assms(2)* **by** *simp*
then show *?thesis*
using *tpsL2' tpsL2''* **by** *simp*
qed

definition *tpsL4* :: *nat* ⇒ *tape list* **where**
tpsL4 t ≡ *tps0*
$[j := (\lfloor idx + step * Suc\ t\rfloor_N,\ 1),$
$1 := nlltape\ (nss\ @\ concat\ (map\ (\lambda i.\ nll\text{-}Psi\ (H * (idx + step * i))\ H\ kappa)\ [0..<Suc\ t])),$
$j + 6 := (\lfloor[]\rfloor,\ 1)]$

lemma *tm4*:
assumes $t < numiter$
and $ttt = 1871 * H \hat{}\ 4 * (nlength\ (idx + step * numiter))^2 + 11 +$
$step * (5 + 2 * nlength\ (idx + step * t + step))$
shows *transforms tm4 (tpsL0 t) ttt (tpsL4 t)*
unfolding *tm4-def*
proof (*tform tps*: *assms(1) H kappa tps0 tpsL2''-def tpsL4-def jk*)
have $idx + step * Suc\ t = idx + step * t + step$
by *simp*
then show $tpsL4\ t = (tpsL2''\ t)[j := (\lfloor idx + step * t + step\rfloor_N,\ 1)]$
unfolding *tpsL4-def tpsL2''-def* **using** *jk* **by** (*simp only*: *list-update-swap[of - j] list-update-overwrite*)
show $ttt = 1871 * H \hat{}\ 4 * (nlength\ (idx + step * numiter))^2 + 11 +$
$step * (5 + 2 * nlength\ (idx + step * t + step))$
using *assms(2)* **.**
qed

lemma *tm4'*:
assumes $t < numiter$
and $ttt = (6 * step + 1882) * H \hat{}\ 4 * (nlength\ (idx + step * numiter))^2$
shows *transforms tm4 (tpsC t) ttt (tpsL4 t)*
proof −
let $?ttt = 1871 * H \hat{}\ 4 * (nlength\ (idx + step * numiter))^2 + 11 +$
$step * (5 + 2 * nlength\ (idx + step * t + step))$
have $idx + step * t + step \le idx + step * numiter$
using *assms(1)*
by (*metis Suc-le-eq add.assoc add.commute add-le-cancel-left add-mult-distrib2 mult-le-mono2 mult-numeral-1-right numerals(1) plus-1-eq-Suc*)
then have $?ttt \le 1871 * H \hat{}\ 4 * (nlength\ (idx + step * numiter))^2 + 11 +$
$step * (5 + 2 * nlength\ (idx + step * numiter))$
using *nlength-mono* **by** *simp*
also have $... = 1871 * H \hat{}\ 4 * (nlength\ (idx + step * numiter))^2 + 11 +$
$step * 5 + step * 2 * nlength\ (idx + step * numiter)$
by *algebra*
also have $... \le 1871 * H \hat{}\ 4 * (nlength\ (idx + step * numiter))^2 + 11 +$
$step * 5 + step * 2 * nlength\ (idx + step * numiter)\hat{}2$
by (*simp add*: *linear-le-pow*)
also have $... \le 1871 * H \hat{}\ 4 * (nlength\ (idx + step * numiter))^2 + 11 +$
$step * 5 + step * H\hat{}4 * nlength\ (idx + step * numiter)\hat{}2$
proof −
have $2 < H$
using *H* **by** *simp*
then have $2 < H \hat{}\ 4$
using *linear-le-pow* **by** (*meson le-trans not-less zero-less-numeral*)
then show *?thesis*

by *simp*
qed
also have ... = $(step + 1871) * H \widehat{\ } 4 * (nlength\ (idx + step * numiter))^2 + (11 + step * 5)$
by *algebra*
also have ... $\leq (step + 1871) * H \widehat{\ } 4 * (nlength\ (idx + step * numiter))^2 + (11 + step * 5) * H \widehat{\ } 4 * (nlength\ (idx + step * numiter))\widehat{\ }2$
proof −
have $H \widehat{\ } 4 * (nlength\ (idx + step * numiter))\widehat{\ }2 > 0$
using *step assms*(*1*) *nlength-0 H* **by** *auto*
then show *?thesis*
by (*smt* (*verit*) *One-nat-def Suc-leI add-mono-thms-linordered-semiring*(*2*) *mult.assoc mult-numeral-1-right nat-mult-le-cancel-disj numeral-code*(*1*))
qed
also have ... = $(6 * step + 1882) * H \widehat{\ } 4 * (nlength\ (idx + step * numiter))^2$
by *algebra*
finally have $?ttt \leq (6 * step + 1882) * H \widehat{\ } 4 * (nlength\ (idx + step * numiter))^2$ **.**
then have *transforms tm4* (*tpsL0 t*) *ttt* (*tpsL4 t*)
using *tm4 assms transforms-monotone* **by** *blast*
then show *?thesis*
using *tpsL0-eq-tpsC assms*(*1*) **by** *simp*
qed

lemma *tm4''* [*transforms-intros*]:
assumes $t < numiter$
and $ttt = (6 * step + 1882) * H \widehat{\ } 4 * (nlength\ (idx + step * numiter))^2$
shows *transforms tm4* (*tpsC t*) *ttt* (*tpsL* (*Suc t*))
proof −
have *tpsL4 t* = *tpsL* (*Suc t*)
using *tpsL4-def tpsL-def jk tps0 list-update-id*[*of tps0 j* + *6*]
by (*simp add*: *list-update-swap*)
then show *?thesis*
using *tm4' assms* **by** *metis*
qed

lemma *tmL*:
assumes $ttt = Suc\ numiter * (9 + (6 * step + 1885) * (H \widehat{\ } 4 * (nlength\ (idx + step * numiter))^2))$
and *nn* = *numiter*
shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsC nn*)
unfolding *tmL-def*
proof (*tform tps*: *assms*(*2*))
let $?ttt = (6 * step + 1882) * H \widehat{\ } 4 * (nlength\ (idx + step * numiter))^2$
show $\bigwedge t.\ t < nn \Longrightarrow read\ (tpsC\ t)\ !\ (j + 3) = \Box$
using *assms*(*2*) *tpsC-def jk read-ncontents-eq-0* **by** *simp*
show $read\ (tpsC\ nn)\ !\ (j + 3) \neq \Box$
using *assms*(*2*) *tpsC-def jk read-ncontents-eq-0* **by** *simp*
show $nn * (3 * nlength\ (idx + step * numiter) + 7 + ?ttt + 2) + (3 * nlength\ (idx + step * numiter) + 7) + 1 \leq ttt$
(**is** $?lhs \leq ttt$)
proof −
let $?g = H \widehat{\ } 4 * (nlength\ (idx + step * numiter))^2$
have $nlength\ (idx + step * numiter) \leq nlength\ (idx + step * numiter)\widehat{\ }2$
using *linear-le-pow* **by** *simp*
moreover have $H \widehat{\ } 4 > 0$
using *H* **by** *simp*
ultimately have $*$: $nlength\ (idx + step * numiter) \leq ?g$
by (*metis ab-semigroup-mult-class.mult-ac*(*1*) *le-square mult.left-commute nat-mult-le-cancel-disj power2-eq-square*)
have $?lhs = numiter * (3 * nlength\ (idx + step * numiter) + 9 + ?ttt) + 3 * nlength\ (idx + step * numiter) + 8$
using *assms*(*2*) **by** *simp*
also have ... $\leq numiter * (3 * nlength\ (idx + step * numiter) + 9 + ?ttt) + 3 * ?g + 8$
using $*$ **by** *simp*
also have ... $\leq numiter * (3 * ?g + 9 + (6 * step + 1882) * ?g) + 3 * ?g + 8$
using $*$ **by** *simp*

also have ... = *numiter* ∗ (*9* + (*6* ∗ *step* + *1885*) ∗ *?g*) + *3* ∗ *?g* + *8*
  by *algebra*
also have ... ≤ *numiter* ∗ (*9* + (*6* ∗ *step* + *1885*) ∗ *?g*) + (*6* ∗ *step* + *1885*) ∗ *?g* + *8*
  by *simp*
also have ... ≤ *numiter* ∗ (*9* + (*6* ∗ *step* + *1885*) ∗ *?g*) + (*9* + (*6* ∗ *step* + *1885*) ∗ *?g*)
  by *simp*
also have ... = *Suc numiter* ∗ (*9* + (*6* ∗ *step* + *1885*) ∗ *?g*)
  by *simp*
finally show *?thesis*
  using *assms*(*1*) **by** *simp*
qed
qed

lemma *tmL′*:
  assumes *ttt* = *Suc numiter* ∗ (*9* + (*6* ∗ *step* + *1885*) ∗ (*H* ^ *4* ∗ (*nlength* (*idx* + *step* ∗ *numiter*))^2))
  shows *transforms tmL tps0 ttt* (*tpsC numiter*)
  using *assms tmL tpsL-eq-tps0* **by** *simp*

end

end

lemma *transforms-tm-PHI345I*:
  fixes *j* :: *tapeidx*
  fixes *tps tps′* :: *tape list* **and** *ttt step numiter H k idx kappa* :: *nat* **and** *nss* :: *nat list list*
  assumes *length tps* = *k* **and** *1* < *j* **and** *j* + *8* < *k*
    and *H* ≥ *3*
    and *kappa* ≤ *H*
    and *step* > *0*
  assumes
    tps ! *1* = *nlltape nss*
    tps ! *j* = (⌊*idx*⌋$_N$, *1*)
    tps ! (*j* + *1*) = (⌊*H*⌋$_N$, *1*)
    tps ! (*j* + *2*) = (⌊*kappa*⌋$_N$, *1*)
    tps ! (*j* + *3*) = (⌊[]⌋, *1*)
    tps ! (*j* + *4*) = (⌊[]⌋, *1*)
    tps ! (*j* + *5*) = (⌊[]⌋, *1*)
    tps ! (*j* + *6*) = (⌊[]⌋, *1*)
    tps ! (*j* + *7*) = (⌊[]⌋, *1*)
    tps ! (*j* + *8*) = (⌊*idx* + *step* ∗ *numiter*⌋$_N$, *1*)
  assumes *ttt* = *Suc numiter* ∗ (*9* + (*6* ∗ *step* + *1885*) ∗ (*H* ^ *4* ∗ (*nlength* (*idx* + *step* ∗ *numiter*))^2))
  assumes *tps′* = *tps*
    [*j* := (⌊*idx* + *step* ∗ *numiter*⌋$_N$, *1*),
     1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*idx* + *step* ∗ *i*)) *H kappa*) [*0*..<*numiter*])),
     j + *3* := (⌊*1*⌋$_N$, *1*)]
  shows *transforms* (*tm-PHI345 step j*) *tps ttt tps′*
proof −
  interpret *loc*: *turing-machine-PHI345 step j* **.**
  show *?thesis*
    using *assms loc.tmL′ loc.tpsC-def loc.tmL-eq-tm-PHI345* **by** *simp*
qed

7.6.5 A Turing machine for Φ_6

The next Turing machine expects a symbol sequence *zs* on input tape 0, the number 2 on tape *j*, and a number *H* on tape $j+1$. It appends to the CNF formula on tape 1 the formula $\bigwedge_{i=0}^{|zs|-1} \Psi(\gamma_{2+2i}, z_i)$, where z_i is 2 or 3 if zs_i is **0** or **1**, respectively.

definition *tm-PHI6* :: *tapeidx* ⇒ *machine* **where**
  tm-PHI6 j ≡
    WHILE [] ; λ*rs*. *rs* ! *0* ≠ □ *DO*
      IF λ*rs*. *rs* ! *0* = **0** *THEN*
        tm-setn (*j* + *2*) *2*
      ELSE

tm-setn $(j + 2)$ *3*
ENDIF ;;
tm-Psigamma j ;;
tm-extendl-erase 1 $(j + 6)$;;
tm-setn $(j + 2)$ *0* ;;
tm-right 0 ;;
tm-plus-const 2 j
DONE

lemma *tm-PHI6-tm*:
assumes $G \geq 6$ **and** $0 < j$ **and** $j + 7 < k$
shows *turing-machine k G* (*tm-PHI6 j*)
unfolding *tm-PHI6-def*
using *assms tm-Psigamma-tm tm-extendl-erase-tm tm-plus-const-tm*
turing-machine-loop-turing-machine turing-machine-branch-turing-machine
tm-right-tm tm-setn-tm Nil-tm
by *simp*

locale *turing-machine-PHI6* =
fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *IF* $\lambda rs.\ rs\ !\ 0 = \mathbf{0}$ *THEN tm-setn* $(j + 2)$ *2 ELSE tm-setn* $(j + 2)$ *3 ENDIF*
definition *tm2* ≡ *tm1* ;; *tm-Psigamma j*
definition *tm3* ≡ *tm2* ;; *tm-extendl-erase 1* $(j + 6)$
definition *tm4* ≡ *tm3* ;; *tm-setn* $(j + 2)$ *0*
definition *tm5* ≡ *tm4* ;; *tm-right 0*
definition *tm6* ≡ *tm5* ;; *tm-plus-const 2 j*
definition *tmL* ≡ *WHILE* [] ; $\lambda rs.\ rs\ !\ 0 \neq \Box$ *DO tm6 DONE*

lemma *tmL-eq-tm-PHI6*: *tmL* = *tm-PHI6 j*
using *tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def tm-PHI6-def tmL-def* **by** *simp*

context
fixes *tps0* :: *tape list* **and** *k H* :: *nat* **and** *zs* :: *symbol list* **and** *nss* :: *nat list list*
assumes *jk*: *length tps0* = k $1 < j$ $j + 7 < k$
and *H*: $H \geq 3$
and *zs*: *bit-symbols zs*
assumes *tps0*:
tps0 ! *0* = (⌊*zs*⌋, *1*)
tps0 ! *1* = *nlltape nss*
tps0 ! *j* = $(\lfloor 2\rfloor_N, 1)$
tps0 ! $(j + 1) = (\lfloor H\rfloor_N, 1)$
tps0 ! $(j + 2) = (\lfloor 0\rfloor_N, 1)$
tps0 ! $(j + 3)$ = (⌊[]⌋, *1*)
tps0 ! $(j + 4)$ = (⌊[]⌋, *1*)
tps0 ! $(j + 5)$ = (⌊[]⌋, *1*)
tps0 ! $(j + 6)$ = (⌊[]⌋, *1*)
tps0 ! $(j + 7)$ = (⌊[]⌋, *1*)
begin

lemma *H0*: $H > 0$
using *H* **by** *simp*

lemma *H-mult*: $x \leq H * x$ $x \leq x * H$
using *H* **by** *simp-all*

definition *tpsL* :: *nat* ⇒ *tape list* **where**
tpsL t ≡ *tps0*
[*0* := (⌊*zs*⌋, *Suc t*),
j := $(\lfloor 2 + 2 * t\rfloor_N, 1)$,
1 := *nlltape* (*nss* @ *concat* (*map* ($\lambda i.$ *nll-Psi* $(H * (2 + 2 * i))$ *H* (*zs* ! *i*)) [*0*..<*t*]))]

lemma *tpsL-eq-tps0*: *tpsL 0 = tps0*
proof −
  have (⌊*zs*⌋, *1*) = (⌊*zs*⌋, *Suc 0*)
    by *simp*
  moreover have *nlltape* (*concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*2* + *2* ∗ *i*)) *H* (*zs* ! *i*)) [*0*..<*0*])) = (⌊[]⌋, *1*)
    using *nllcontents-Nil* **by** *simp*
  moreover have *2* = *Suc* (*Suc 0*)
    by *simp*
  ultimately show *?thesis*
    using *tpsL-def tps0 jk* **by** *simp* (*metis One-nat-def Suc-1 list-update-id*)
qed

definition *tpsL1* :: *nat* ⇒ *tape list* **where**
  tpsL1 t ≡ *tps0*
    [*0* := (⌊*zs*⌋, *Suc t*),
     j := (⌊*2* + *2* ∗ *t*⌋$_N$, *1*),
     j + *2* := (⌊*zs* ! *t*⌋$_N$, *1*),
     1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*2* + *2* ∗ *i*)) *H* (*zs* ! *i*)) [*0*..<*t*]))]

lemma *tm1* [*transforms-intros*]:
  assumes *ttt* = *16* **and** *t* < *length zs*
  shows *transforms tm1* (*tpsL t*) *ttt* (*tpsL1 t*)
  unfolding *tm1-def*
proof (*tform tps*: *tpsL-def tps0 jk*)
  have ∗: *read* (*tpsL t*) ! *0* = *zs* ! *t*
    using *jk tpsL-def tapes-at-read*′[*of 0 tpsL t*] *assms*(*2*) **by** *simp*
  show *read* (*tpsL t*) ! *0* = **0** ⟹ *tpsL1 t* = (*tpsL t*)[*j* + *2* := (⌊*2*⌋$_N$, *1*)]
    using ∗ *tpsL-def tpsL1-def jk* **by** (*simp add*: *list-update-swap*)
  show *tpsL1 t* = (*tpsL t*)[*j* + *2* := (⌊*3*⌋$_N$, *1*)] **if** *read* (*tpsL t*) ! □ ≠ **0**
  proof −
    have *zs* ! *t* = **1**
      using ∗ *that zs assms*(*2*) **by** *auto*
    then show *?thesis*
      using *tpsL-def tpsL1-def jk* **by** (*simp add*: *list-update-swap*)
  qed
  show *10* + *2* ∗ *nlength 0* + *2* ∗ *nlength 2* + *2* ≤ *ttt*
    using *nlength-0 nlength-2 assms*(*1*) **by** *simp*
  show *10* + *2* ∗ *nlength 0* + *2* ∗ *nlength 3* + *1* ≤ *ttt*
    using *nlength-0 nlength-3 assms*(*1*) **by** *simp*
qed

definition *tpsL2* :: *nat* ⇒ *tape list* **where**
  tpsL2 t ≡ *tps0*
    [*0* := (⌊*zs*⌋, *Suc t*),
     j := (⌊*2* + *2* ∗ *t*⌋$_N$, *1*),
     j + *2* := (⌊*zs* ! *t*⌋$_N$, *1*),
     j + *6* := (⌊*nll-Psi* (*2* ∗ *H* + *2* ∗ *t* ∗ *H*) *H* (*zs* ! *t*)⌋$_{NLL}$, *Suc 0*),
     1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*2* + *2* ∗ *i*)) *H* (*zs* ! *i*)) [*0*..<*t*]))]

lemma *tm2* [*transforms-intros*]:
  assumes *ttt* = *16* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc* (*Suc* (*Suc* (*2* ∗ *t*)))))^2
    and *t* < *length zs*
  shows *transforms tm2* (*tpsL t*) *ttt* (*tpsL2 t*)
  unfolding *tm2-def*
proof (*tform tps*: *assms*(*2*) *tps0 H tpsL1-def tpsL2-def jk time*: *assms*(*1*))
  show *zs* ! *t* ≤ *H*
    using *assms*(*2*) *H zs* **by** *auto*
qed

definition *tpsL3* :: *nat* ⇒ *tape list* **where**
  tpsL3 t ≡ *tps0*
    [*0* := (⌊*zs*⌋, *Suc t*),
     j := (⌊*2* + *2* ∗ *t*⌋$_N$, *1*),

```
    j + 2 := (⌊zs ! t⌋N, 1),
    j + 6 := (⌊[]⌋, 1),
    1 := nlltape (nss @ concat (map (λi. nll-Psi (2 ∗ H + H ∗ (2 ∗ i)) H (zs ! i)) [0..<Suc t]))]

lemma tm3 [transforms-intros]:
  assumes ttt = 27 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc (2 ∗ t)))))² +
      4 ∗ nlllength (nll-Psi (2 ∗ H + 2 ∗ t ∗ H) H (zs ! t))
    and t < length zs
  shows transforms tm3 (tpsL t) ttt (tpsL3 t)
  unfolding tm3-def
proof (tform tps: assms(2) tps0 H tpsL2-def tpsL3-def jk time: assms(1))
  have nll-Psi (2 ∗ H + H ∗ (2 ∗ t)) H = nll-Psi (2 ∗ H + 2 ∗ t ∗ H) H
    by (simp add: mult.commute)
  then have (nss @ concat (map (λi. nll-Psi (2 ∗ H + H ∗ (2 ∗ i)) H (zs ! i)) [0..<t])) @
          nll-Psi (2 ∗ H + 2 ∗ t ∗ H) H (zs ! t) =
        nss @ concat (map (λi. nll-Psi (2 ∗ H + H ∗ (2 ∗ i)) H (zs ! i)) [0..<Suc t])
    using assms(2) by simp
  then show tpsL3 t = (tpsL2 t)
      [1 := nlltape
          ((nss @ concat (map (λi. nll-Psi (2 ∗ H + H ∗ (2 ∗ i)) H (zs ! i)) [0..<t])) @
            nll-Psi (2 ∗ H + 2 ∗ t ∗ H) H (zs ! t)),
       j + 6 := (⌊[]⌋, 1)]
    unfolding tpsL3-def tpsL2-def jk by (simp add: list-update-swap)
qed

definition tpsL4 :: nat ⇒ tape list where
  tpsL4 t ≡ tps0
    [0 := (⌊zs⌋, Suc t),
     j := (⌊2 + 2 ∗ t⌋N, 1),
     j + 2 := (⌊0⌋N, 1),
     j + 6 := (⌊[]⌋, 1),
     1 := nlltape (nss @ concat (map (λi. nll-Psi (2 ∗ H + H ∗ (2 ∗ i)) H (zs ! i)) [0..<Suc t]))]

lemma tm4 [transforms-intros]:
  assumes ttt = 41 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc (2 ∗ t)))))² +
      4 ∗ nlllength (nll-Psi (2 ∗ H + 2 ∗ t ∗ H) H (zs ! t))
    and t < length zs
  shows transforms tm4 (tpsL t) ttt (tpsL4 t)
  unfolding tm4-def
proof (tform tps: assms(2) tpsL3-def tpsL4-def jk)
  have zs ! t = 2 ∨ zs ! t = 3
    using zs assms(2) by auto
  then show ttt = 27 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc (2 ∗ t)))))² +
      4 ∗ nlllength (nll-Psi (2 ∗ H + 2 ∗ t ∗ H) H (zs ! t)) +
      (10 + 2 ∗ nlength (zs ! t) + 2 ∗ nlength 0)
    using nlength-2 nlength-3 assms(1) by auto
qed

definition tpsL5 :: nat ⇒ tape list where
  tpsL5 t ≡ tps0
    [0 := (⌊zs⌋, Suc (Suc t)),
     j := (⌊2 + 2 ∗ t⌋N, 1),
     j + 2 := (⌊0⌋N, 1),
     j + 6 := (⌊[]⌋, 1),
     1 := nlltape (nss @ concat (map (λi. nll-Psi (2 ∗ H + H ∗ (2 ∗ i)) H (zs ! i)) [0..<Suc t]))]

lemma tm5 [transforms-intros]:
  assumes ttt = 42 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc (2 ∗ t)))))² +
      4 ∗ nlllength (nll-Psi (2 ∗ H + 2 ∗ t ∗ H) H (zs ! t))
    and t < length zs
  shows transforms tm5 (tpsL t) ttt (tpsL5 t)
  unfolding tm5-def
proof (tform tps: assms(2) tps0 H tpsL4-def tpsL5-def jk time: assms(1))
```

have *neq*: $0 \neq j$
  using *jk* **by** *simp*
have *tpsL4 t ! 0* = (⌊*zs*⌋, *Suc t*)
  using *tpsL4-def jk* **by** *simp*
then show *tpsL5 t* = (*tpsL4 t*)[*0* := *tpsL4 t ! 0* |+| *1*]
  unfolding *tpsL5-def tpsL4-def jk* **by** (*simp add*: *list-update-swap*[*OF neq*] *list-update-swap*[*of - 0*])
qed

definition *tpsL6* :: *nat* ⇒ *tape list* **where**
  tpsL6 t ≡ *tps0*
    [*0* := (⌊*zs*⌋, *Suc* (*Suc t*)),
     j := (⌊*2* + *2* ∗ (*Suc t*)⌋$_N$, *1*),
     j + *2* := (⌊*0*⌋$_N$, *1*),
     j + *6* := (⌊[]⌋, *1*),
     1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*2* ∗ *H* + *H* ∗ (*2* ∗ *i*)) *H* (*zs* ! *i*)) [*0*..<*Suc t*]))]

lemma *tm6*:
  assumes *ttt* = *52* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc* (*Suc* (*Suc* (*2* ∗ *t*)))))^2 +
      4 ∗ *nlllength* (*nll-Psi* (*2* ∗ *H* + *2* ∗ *t* ∗ *H*) *H* (*zs* ! *t*)) +
      4 ∗ *nlength* (*4* + *2* ∗ *t*)
    and *t* < *length zs*
  shows *transforms tm6* (*tpsL t*) *ttt* (*tpsL6 t*)
  unfolding *tm6-def*
proof (*tform tps*: *tps0 H tpsL5-def tpsL6-def jk assms*(*2*))
  show *tpsL6 t* = (*tpsL5 t*)[*j* := (⌊*Suc* (*Suc* (*2* ∗ *t*)) + *2*⌋$_N$, *1*)]
    using *tpsL6-def tpsL5-def jk* **by** (*simp add*: *list-update-swap*)
  have ∗: *4* + *2* ∗ *t* = *Suc* (*Suc* (*Suc* (*Suc* (*2* ∗ *t*))))
    by *simp*
  then show *ttt* = *42* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc* (*Suc* (*Suc* (*2* ∗ *t*)))))^2 +
      4 ∗ *nlllength* (*nll-Psi* (*2* ∗ *H* + *2* ∗ *t* ∗ *H*) *H* (*zs* ! *t*)) +
      2 ∗ (*5* + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *t*)) + *2*))
    using *assms*(*1*) ∗ **by** *simp*
qed

lemma *tpsL6-eq-tpsL*: *tpsL6 t* = *tpsL* (*Suc t*)
proof −
  have *tpsL* (*Suc t*) ! (*j* + *6*) = (⌊[]⌋, *1*)
    using *tpsL-def tps0 jk* **by** *simp*
  then have *tpsL* (*Suc t*) = (*tpsL* (*Suc t*))[*j* + *6* := (⌊[]⌋, *1*)]
    using *list-update-id* **by** *metis*
  moreover have *tpsL* (*Suc t*) ! (*j* + *2*) = (⌊*0*⌋$_N$, *1*)
    using *tpsL-def tps0 jk canrepr-0* **by** *simp*
  ultimately have *tpsL* (*Suc t*) = (*tpsL* (*Suc t*))[*j* + *6* := (⌊[]⌋, *1*), *j* + *2* := (⌊*0*⌋$_N$, *1*)]
    using *list-update-id* **by** *metis*
  moreover have *tpsL6 t* = (*tpsL* (*Suc t*))[*j* + *6* := (⌊[]⌋, *1*), *j* + *2* := (⌊*0*⌋$_N$, *1*)]
    unfolding *tpsL6-def tpsL-def* **by** (*simp add*: *list-update-swap*)
  ultimately show *?thesis*
    by *simp*
qed

lemma *tm6'*:
  assumes *ttt* = *133648* ∗ *H* ^ *6* ∗ *length zs*^*2*
    and *t* < *length zs*
  shows *transforms tm6* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
proof −
  have ∗∗: *Suc* (*2* ∗ *length zs*)^*2* ≤ *9* ∗ *length zs* ^ *2*
  proof −
    have *Suc* (*2* ∗ *length zs*)^*2* = *1* + *2* ∗ *2* ∗ *length zs* + (*2* ∗ *length zs*)^*2*
      by (*metis add.commute add-Suc-shift mult.assoc nat-mult-1-right one-power2 plus-1-eq-Suc power2-sum*)
    also have ... = *1* + *4* ∗ *length zs* + *4* ∗ *length zs*^*2*
      by *simp*
    also have ... ≤ *length zs* + *4* ∗ *length zs* + *4* ∗ *length zs*^*2*
      using *assms*(*2*) **by** *simp*

also have ... = *5 ∗ length zs + 4 ∗ length zs^2*
  by *simp*
also have ... ≤ *5 ∗ length zs^2 + 4 ∗ length zs^2*
  using *linear-le-pow* **by** *simp*
also have ... = *9 ∗ length zs^2*
  by *simp*
finally show *?thesis*
  by *simp*
qed

have ∗: *t ≤ length zs − 1*
  using *assms*(*2*) **by** *simp*

let *?ttt = 52 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc (2 ∗ t)))))*^{2} *+*
  4 ∗ nlllength (nll-Psi (2 ∗ H + 2 ∗ t ∗ H) H (zs ! t)) +
  4 ∗ nlength (4 + 2 ∗ t)
have *?ttt = 52 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc (2 ∗ t)))))*^{2} *+*
  4 ∗ nlllength (nll-Psi (2 ∗ H + H ∗ (2 ∗ t)) H (zs ! t)) +
  4 ∗ nlength (4 + 2 ∗ t)
 by (*simp add*: *mult.commute*)
also have ... ≤ *52 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (Suc (Suc (2 ∗ t)))))*^{2} *+*
  4 ∗ nlllength (nll-Psi (2 ∗ H + H ∗ (2 ∗ t)) H (zs ! t)) +
  4 ∗ nlength (2 + 2 ∗ length zs)
 using *nlength-mono assms*(*2*) **by** *simp*
also have ... ≤ *52 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (2 ∗ length zs)))*^{2} *+*
  4 ∗ nlllength (nll-Psi (2 ∗ H + H ∗ (2 ∗ t)) H (zs ! t)) +
  4 ∗ nlength (2 + 2 ∗ length zs)
 using *H4-nlength H assms*(*2*) **by** *simp*
also have ... ≤ *52 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (2 ∗ length zs)))*^{2} *+*
  4 ∗ H ∗ (3 + nlength (3 ∗ H + H ∗ (2 ∗ (length zs − 1)))) +
  4 ∗ nlength (2 + 2 ∗ length zs)
 using *nlllength-nll-Psi-le′*[*of 2 ∗ H + H ∗ (2 ∗ t) 2 ∗ H + H ∗ (2 ∗ (length zs − 1)) H*] ∗
 by *simp*
also have ... = *52 + 1851 ∗ H ^ 4 ∗ (nlength (Suc (2 ∗ length zs)))*^{2} *+*
  4 ∗ H ∗ (3 + nlength (H ∗ (1 + 2 ∗ length zs))) + 4 ∗ nlength (2 + 2 ∗ length zs)
proof −
 have *3 ∗ H + H ∗ (2 ∗ (length zs − 1)) = H ∗ (3 + 2 ∗ (length zs − 1))*
  by *algebra*
 also have ... = *H ∗ (3 + 2 ∗ length zs − 2)*
  using *assms*(*2*)
  by (*metis Nat.add-diff-assoc Suc-pred add-mono-thms-linordered-semiring*(*1*) *le-numeral-extra*(*4*)
   length-greater-0-conv list.size(*3*) *mult-2 nat-1-add-1 not-less-zero plus-1-eq-Suc*
   right-diff-distrib′ trans-le-add1)
 also have ... = *H ∗ (1 + 2 ∗ length zs)*
  by *simp*
 finally have *3 ∗ H + H ∗ (2 ∗ (length zs − 1)) = H ∗ (1 + 2 ∗ length zs)* **.**
 then show *?thesis*
  by *metis*
qed
also have ... ≤ *52 + 1851 ∗ H ^ 4 ∗ (3 + nlength (Suc (2 ∗ length zs)))*^{2} *+*
  4 ∗ H ∗ (3 + nlength (H ∗ (1 + 2 ∗ length zs))) + 4 ∗ nlength (2 + 2 ∗ length zs)
 by *simp*
also have ... ≤ *52 + 1851 ∗ H ^ 4 ∗ (3 + nlength (H ∗ Suc (2 ∗ length zs)))*^{2} *+*
  4 ∗ H ∗ (3 + nlength (H ∗ Suc (2 ∗ length zs))) + 4 ∗ nlength (2 + 2 ∗ length zs)
proof −
 have *Suc (2 ∗ length zs) ≤ H ∗ Suc (2 ∗ length zs)*
  using *H-mult* **by** *blast*
 then have *nlength (Suc (2 ∗ length zs))^2 ≤ nlength (H ∗ Suc (2 ∗ length zs))^2*
  using *nlength-mono* **by** *simp*
 then show *?thesis*
  by *simp*
qed
also have ... ≤ *52 + 1851 ∗ H ^ 4 ∗ (3 + nlength (H ∗ Suc (2 ∗ length zs)))*^{2} *+*

```
    4 * H * (3 + nlength (H * Suc (2 * length zs)))^2 + 4 * nlength (2 + 2 * length zs)
   using linear-le-pow by simp
  also have ... ≤ 52 + 1851 * H ^ 4 * (3 + nlength (H * Suc (2 * length zs)))² +
    4 * H^4 * (3 + nlength (H * Suc (2 * length zs)))^2 + 4 * nlength (2 + 2 * length zs)
   using linear-le-pow by simp
  also have ... = 52 + 1855 * H ^ 4 * (3 + nlength (H * Suc (2 * length zs)))² +
    4 * nlength (2 + 2 * length zs)
   by simp
  also have ... ≤ 52 + 1855 * H ^ 4 * (3 + nlength (H * Suc (2 * length zs)))² +
    4 * nlength (H * Suc (2 * length zs))
  proof −
   have 2 + 2 * m ≤ H * Suc (2 * m) if m > 0 for m
    using H H-mult(2) by (metis Suc-leD add-mono eval-nat-numeral(3) mult.commute mult-Suc-right)
   then have 2 + 2 * length zs ≤ H * Suc (2 * length zs)
    using assms(2) by (metis less-nat-zero-code not-gr-zero)
   then show ?thesis
    using nlength-mono by simp
  qed
  also have ... ≤ 52 + 1855 * H ^ 4 * (3 + H * Suc (2 * length zs))² +
    4 * nlength (H * Suc (2 * length zs))
   using nlength-le-n nlength-mono by simp
  also have ... ≤ 52 + 1855 * H ^ 4 * (3 + H * Suc (2 * length zs))² +
    4 * (H * Suc (2 * length zs))
   using nlength-le-n nlength-mono by (meson add-left-mono mult-le-cancel1)
  also have ... = 52 + 1855 * H ^ 4 * (3^2 + 2 * 3 * H * Suc (2 * length zs) + H^2 * Suc (2 * length zs)^2)
+
    4 * (H * Suc (2 * length zs))
   by algebra
  also have ... ≤ 52 + 1855 * H ^ 4 * (72 * H^2 * length zs^2) + 4 * (H * Suc (2 * length zs))
  proof −
   have 3^2 + 2 * 3 * H * Suc (2 * length zs) + H^2 * Suc (2 * length zs)^2 =
     9 + 6 * H * Suc (2 * length zs) + H^2 * Suc (2 * length zs)^2
    by simp
   also have ... ≤ 9 + 6 * H^2 * Suc (2 * length zs) + H^2 * Suc (2 * length zs)^2
    using H linear-le-pow by (simp add: add-mono)
   also have ... ≤ 9 + 6 * H^2 * Suc (2 * length zs)^2 + H^2 * Suc (2 * length zs)^2
    using linear-le-pow by (meson add-le-mono1 add-left-mono le-refl mult-le-mono zero-less-numeral)
   also have ... = 9 + 7 * H^2 * Suc (2 * length zs)^2
    by simp
   also have ... ≤ 9 + 7 * H^2 * 9 * length zs^2
    using ** by simp
   also have ... = 9 + 63 * H^2 * length zs^2
    by simp
   also have ... ≤ 9 * H^2 * length zs^2 + 63 * H^2 * length zs^2
    using assms(2) H by simp
   also have ... = 72 * H^2 * length zs^2
    by simp
   finally have 3^2 + 2 * 3 * H * Suc (2 * length zs) + H^2 * Suc (2 * length zs)^2 ≤ 72 * H^2 * length
zs^2 .
   then show ?thesis
    using add-le-mono le-refl mult-le-mono2 by presburger
  qed
  also have ... = 52 + 133560 * H ^ 6 * length zs^2 + 4 * (H * Suc (2 * length zs))
   by simp
  also have ... ≤ 52 + 133560 * H ^ 6 * length zs^2 + 4 * (H * 9 * length zs ^ 2)
   using ** by (smt (verit) add-left-mono mult.assoc mult-le-cancel1 power2-nat-le-imp-le)
  also have ... = 52 + 133560 * H ^ 6 * length zs^2 + 36 * H * length zs ^ 2
   by simp
  also have ... ≤ 52 + 133560 * H ^ 6 * length zs^2 + 36 * H^6 * length zs ^ 2
   using linear-le-pow by simp
  also have ... = 52 + 133596 * H ^ 6 * length zs^2
   by simp
  also have ... ≤ 52 * H ^ 6 * length zs^2 + 133596 * H ^ 6 * length zs^2
```

```
    using H assms(2) by simp
  also have ... = 133648 * H ^ 6 * length zs^2
    by simp
  finally have ?ttt ≤ 133648 * H ^ 6 * length zs^2 .
  then have transforms tm6 (tpsL t) ttt (tpsL6 t)
    using tm6 assms transforms-monotone by blast
  then show ?thesis
    using tpsL6-eq-tpsL by simp
qed

lemma tmL:
  assumes ttt = 133650 * H ^ 6 * length zs ^ 3 + 1
  shows transforms tmL (tpsL 0) ttt (tpsL (length zs))
  unfolding tmL-def
proof (tform)
  let ?t = 133648 * H ^ 6 * length zs^2
  show ⋀i. i < length zs ⟹ transforms tm6 (tpsL i) ?t (tpsL (Suc i))
    using tm6' by simp
  have *: read (tpsL t) ! 0 = ⌊zs⌋ (Suc t) for t
    using jk tapes-at-read'[symmetric, of 0 tpsL t] by (simp add: tpsL-def)
  show read (tpsL t) ! 0 ≠ □ if t < length zs for t
  proof -
    have read (tpsL t) ! 0 = zs ! t
      using that * by simp
    then show ?thesis
      using that zs by auto
  qed
  show ¬ read (tpsL (length zs)) ! 0 ≠ □
    using * by simp
  show length zs * (?t + 2) + 1 ≤ ttt
  proof (cases length zs = 0)
    case True
    then show ?thesis
      using assms(1) by simp
  next
    case False
    then have 1 ≤ H^6 * length zs ^ 2
      using H linear-le-pow by (simp add: Suc-leI)
    then have ?t + 2 ≤ 133650 * H ^ 6 * length zs^2
      by simp
    then have length zs * (?t + 2) ≤ length zs * 133650 * H ^ 6 * length zs^2
      by simp
    then have length zs * (?t + 2) ≤ 133650 * H ^ 6 * length zs^3
      by (simp add: power2-eq-square power3-eq-cube)
    then show ?thesis
      using assms(1) by simp
  qed
qed

lemma tmL':
  assumes ttt = 133650 * H ^ 6 * length zs ^ 3 + 1
  shows transforms tmL tps0 ttt (tpsL (length zs))
  using assms tmL tpsL-eq-tps0 by simp

end

end

lemma transforms-tm-PHI6I:
  fixes j :: tapeidx
  fixes tps tps' :: tape list and ttt k H :: nat and zs :: symbol list and nss :: nat list list
  assumes length tps = k and 1 < j and j + 7 < k
    and H ≥ 3
```

and *bit-symbols zs*
assumes
tps ! *1* = *nlltape nss*
tps ! *0* = (⌊*zs*⌋, *1*)
tps ! *j* = (⌊*2*⌋$_N$, *1*)
tps ! (*j* + *1*) = (⌊*H*⌋$_N$, *1*)
tps ! (*j* + *2*) = (⌊*0*⌋$_N$, *1*)
tps ! (*j* + *3*) = (⌊[]⌋, *1*)
tps ! (*j* + *4*) = (⌊[]⌋, *1*)
tps ! (*j* + *5*) = (⌊[]⌋, *1*)
tps ! (*j* + *6*) = (⌊[]⌋, *1*)
tps ! (*j* + *7*) = (⌊[]⌋, *1*)
assumes *tps'* = *tps*
[*0* := (⌊*zs*⌋, *Suc* (*length zs*)),
j := (⌊*2* + *2* ∗ *length zs*⌋$_N$, *1*),
1 := *nlltape* (*nss* @ *concat* (*map* (λ*i*. *nll-Psi* (*H* ∗ (*2* + *2* ∗ *i*)) *H* (*zs* ! *i*)) [*0*..<*length zs*]))]
assumes *ttt* = *133650* ∗ *H* ^ *6* ∗ *length zs* ^ *3* + *1*
shows *transforms* (*tm-PHI6 j*) *tps ttt tps'*
proof −
interpret *loc*: *turing-machine-PHI6 j* **.**
show *?thesis*
using *assms loc.tpsL-def loc.tmL' loc.tmL-eq-tm-PHI6* **by** *simp*
qed

7.6.6 A Turing machine for Φ_7

The next Turing machine expects a number *idx* on tape j, a number H on tape $j+1$, and a number *numiter* on tape $j+6$. It appends to the CNF formula on tape 1 the formula $\bigwedge_{t=0}^{numiter} \Upsilon(\gamma_{idx+2t})$ with $\gamma_i = [iH, (i+1)H)$. This equals Φ_7 if $idx = 2n+4$ and $numiter = p(n)$.

definition *tm-PHI7* :: *tapeidx* ⇒ *machine* **where**
tm-PHI7 j ≡
WHILE [] ; λ*rs*. *rs* ! (*j* + *6*) ≠ □ *DO*
tm-Upsilongamma j ;;
tm-extendl-erase *1* (*j* + *4*) ;;
tm-plus-const *2 j* ;;
tm-decr (*j* + *6*)
DONE

lemma *tm-PHI7-tm*:
assumes *0* < *j* **and** *j* + *6* < *k* **and** *6* ≤ *G* **and** *2* ≤ *k*
shows *turing-machine k G* (*tm-PHI7 j*)
unfolding *tm-PHI7-def*
using *tm-Upsilongamma-tm tm-decr-tm Nil-tm tm-extendl-erase-tm tm-plus-const-tm assms turing-machine-loop-turing-machine*
by *simp*

locale *turing-machine-tm-PHI7* =
fixes *step* :: *nat* **and** *j* :: *tapeidx*
begin

definition *tmL1* ≡ *tm-Upsilongamma j*
definition *tmL2* ≡ *tmL1* ;; *tm-extendl-erase* *1* (*j* + *4*)
definition *tmL4* ≡ *tmL2* ;; *tm-plus-const* *2 j*
definition *tmL5* ≡ *tmL4* ;; *tm-decr* (*j* + *6*)
definition *tmL* ≡ *WHILE* [] ; λ*rs*. *rs* ! (*j* + *6*) ≠ □ *DO tmL5 DONE*

lemma *tmL-eq-tm-PHI7*: *tmL* = *tm-PHI7 j*
unfolding *tmL-def tmL5-def tmL4-def tmL2-def tmL1-def tm-PHI7-def* **by** *simp*

context
fixes *tps0* :: *tape list* **and** *numiter H k idx* :: *nat* **and** *nss* :: *nat list list*
assumes *jk*: *length tps0* = *k* *1* < *j* *j* + *6* < *k*
and *H*: *H* ≥ *3*
assumes *tps0*:

tps0 ! *1* = *nlltape nss*
tps0 ! *j* = $(\lfloor idx \rfloor_N, 1)$
tps0 ! $(j + 1) = (\lfloor H \rfloor_N, 1)$
tps0 ! $(j + 2) = (\lfloor [] \rfloor, 1)$
tps0 ! $(j + 3) = (\lfloor [] \rfloor, 1)$
tps0 ! $(j + 4) = (\lfloor [] \rfloor, 1)$
tps0 ! $(j + 5) = (\lfloor [] \rfloor, 1)$
tps0 ! $(j + 6) = (\lfloor numiter \rfloor_N, 1)$
begin

lemma *nlength-H*: *nlength H* $\geq$ *1*
using *nlength-0 H* **by** (*metis dual-order.trans nlength-1-simp nlength-mono one-le-numeral*)

definition *tpsL* :: *nat* ⇒ *tape list* **where**
tpsL t ≡ *tps0*
$[j := (\lfloor idx + 2 * t \rfloor_N, 1),$
$j + 6 := (\lfloor numiter - t \rfloor_N, 1),$
1 := *nlltape* (*nss* @ *concat* (*map* (λ*t*. *nll-Upsilon* (*idx* + *2* ∗ *t*) *H*) [*0*..<*t*]))]

lemma *tpsL-eq-tps0*: *tpsL 0* = *tps0*
using *tpsL-def tps0* **by** *auto* (*metis list-update-id*)

definition *tpsL1* :: *nat* ⇒ *tape list* **where**
tpsL1 t ≡ *tps0*
$[j := (\lfloor idx + 2 * t \rfloor_N, 1),$
$j + 6 := (\lfloor numiter - t \rfloor_N, 1),$
1 := *nlltape* (*nss* @ *concat* (*map* (λ*t*. *nll-Upsilon* (*idx* + *2* ∗ *t*) *H*) [*0*..<*t*])),
$j + 4 := (\lfloor$*nll-Upsilon* (*idx* + *2* ∗ *t*) *H*$\rfloor_{NLL}, 1)]$

lemma *tmL1* [*transforms-intros*]:
assumes *t* < *numiter*
and $ttt = 205 * H * (nlength\ (idx + 2 * t) + nlength\ H)^2$
shows *transforms tmL1* (*tpsL t*) *ttt* (*tpsL1 t*)
unfolding *tmL1-def* **by** (*tform tps*: *H tps0 tpsL-def tpsL1-def jk time*: *assms*(*2*))

definition *tpsL2* :: *nat* ⇒ *tape list* **where**
tpsL2 t ≡ *tps0*
$[j := (\lfloor idx + 2 * t \rfloor_N, 1),$
$j + 6 := (\lfloor numiter - t \rfloor_N, 1),$
1 := *nlltape* (*nss* @ *concat* (*map* (λ*t*. *nll-Upsilon* (*idx* + *2* ∗ *t*) *H*) [*0*..<*t*]) @ (*nll-Upsilon* (*idx* + *2* ∗ *t*) *H*)),
$j + 4 := (\lfloor [] \rfloor, 1)]$

lemma *tmL2* [*transforms-intros*]:
assumes *t* < *numiter*
and $ttt = 11 + 205 * H * (nlength\ (idx + 2 * t) + nlength\ H)^2 +$
4 ∗ *nlllength* (*nll-Upsilon* (*idx* + *2* ∗ *t*) *H*)
shows *transforms tmL2* (*tpsL t*) *ttt* (*tpsL2 t*)
unfolding *tmL2-def* **by** (*tform tps*: *assms*(*1*) *H tps0 tpsL1-def tpsL2-def jk time*: *assms*(*2*))

definition *tpsL4* :: *nat* ⇒ *tape list* **where**
tpsL4 t ≡ *tps0*
$[j := (\lfloor idx + 2 * Suc\ t \rfloor_N, 1),$
$j + 6 := (\lfloor numiter - t \rfloor_N, 1),$
1 := *nlltape* (*nss* @ *concat* (*map* (λ*t*. *nll-Upsilon* (*idx* + *2* ∗ *t*) *H*) [*0*..<*t*]) @ (*nll-Upsilon* (*idx* + *2* ∗ *t*) *H*)),
$j + 4 := (\lfloor [] \rfloor, 1)]$

lemma *tmL4* [*transforms-intros*]:
assumes *t* < *numiter*
and $ttt = 21 + 205 * H * (nlength\ (idx + 2 * t) + nlength\ H)^2 +$
4 ∗ *nlllength* (*nll-Upsilon* (*idx* + *2* ∗ *t*) *H*) + *4* ∗ *nlength* (*Suc* (*Suc* (*idx* + *2* ∗ *t*)))
shows *transforms tmL4* (*tpsL t*) *ttt* (*tpsL4 t*)
unfolding *tmL4-def*
proof (*tform tps*: *assms*(*1*) *H tps0 tpsL2-def tpsL4-def jk time*: *assms*(*2*))

show *tpsL4 t* = (*tpsL2 t*)[$j := (\lfloor idx + 2 * t + 2\rfloor_N, 1)$]
  using *tpsL4-def tpsL2-def jk* **by** (*simp add*: *list-update-swap*[*of j*])
qed

definition *tpsL5* :: *nat* ⇒ *tape list* **where**
  $tpsL5\ t \equiv tps0$
  $[j := (\lfloor idx + 2 * Suc\ t\rfloor_N, 1),$
  $j + 6 := (\lfloor numiter - Suc\ t\rfloor_N, 1),$
  $1 := nlltape\ (nss\ @\ concat\ (map\ (\lambda t.\ nll\text{-}Upsilon\ (idx + 2 * t)\ H)\ [0..<t])\ @\ (nll\text{-}Upsilon\ (idx + 2 * t)\ H)),$
  $j + 4 := (\lfloor[]\rfloor, 1)]$

lemma *tmL5*:
  assumes $t < numiter$
    and $ttt = 29 + 205 * H * (nlength\ (idx + 2 * t) + nlength\ H)^2 +$
      $4 * nlllength\ (nll\text{-}Upsilon\ (idx + 2 * t)\ H) + 4 * nlength\ (Suc\ (Suc\ (idx + 2 * t))) +$
      $2 * nlength\ (numiter - t)$
  shows *transforms tmL5* (*tpsL t*) *ttt* (*tpsL5 t*)
  unfolding *tmL5-def*
proof (*tform tps*: *assms*(*1*) *H tps0 tpsL4-def tpsL5-def jk time*: *assms*(*2*))
  show *tpsL5 t* = (*tpsL4 t*)[$j + 6 := (\lfloor numiter - t - 1\rfloor_N, 1)$]
    using *tpsL5-def tpsL4-def jk* **by** (*simp add*: *list-update-swap*[*of j+6*])
qed

lemma *tpsL5-eq-tpsL*: *tpsL5 t* = *tpsL* (*Suc t*)
proof −
  define *tps* **where** *tps* = *tps0*
    $[j := (\lfloor idx + 2 * Suc\ t\rfloor_N, 1),$
    $j + 6 := (\lfloor numiter - Suc\ t\rfloor_N, 1),$
    $1 := nlltape\ (nss\ @\ concat\ (map\ (\lambda t.\ nll\text{-}Upsilon\ (idx + 2 * t)\ H)\ [0..<t])\ @\ (nll\text{-}Upsilon\ (idx + 2 * t)\ H))]$
  then have *tps* = *tpsL* (*Suc t*)
    using *tpsL-def jk tps0* **by** *simp*
  moreover have *tps* = *tpsL5 t*
  proof −
    have *tps* ! $(j + 4) = (\lfloor[]\rfloor, 1)$
      using *tps-def jk tps0* **by** *simp*
    then have *tps*[$j + 4 := (\lfloor[]\rfloor, 1)$] = *tps*
      using *list-update-id*[*of - j+4*] **by** *metis*
    then show *?thesis*
      unfolding *tps-def* **using** *tpsL5-def* **by** *simp*
  qed
  ultimately show *?thesis*
    by *simp*
qed

lemma *tmL5′* [*transforms-intros*]:
  assumes $t < numiter$
    and $ttt = 256 * H * (nlength\ (idx + 2 * numiter) + nlength\ H)^2$
  shows *transforms tmL5* (*tpsL t*) *ttt* (*tpsL* (*Suc t*))
proof −
  let $?ttt = 29 + 205 * H * (nlength\ (idx + 2 * t) + nlength\ H)^2 +$
    $4 * nlllength\ (nll\text{-}Upsilon\ (idx + 2 * t)\ H) + 4 * nlength\ (Suc\ (Suc\ (idx + 2 * t))) +$
    $2 * nlength\ (numiter - t)$
  have $?ttt \le 29 + 205 * H * (nlength\ (idx + 2 * t) + nlength\ H)^2 +$
    $4 * nlllength\ (nll\text{-}Upsilon\ (idx + 2 * t)\ H) + 4 * nlength\ (Suc\ (Suc\ (idx + 2 * t))) +$
    $2 * nlength\ numiter$
    using *nlength-mono* **by** *simp*
  also have ... $\le 29 + 205 * H * (nlength\ (idx + 2 * t) + nlength\ H)^2 +$
    $4 * H * (4 + nlength\ (idx + 2 * t) + nlength\ H) + 4 * nlength\ (Suc\ (Suc\ (idx + 2 * t))) +$
    $2 * nlength\ numiter$
    using *nlllength-nll-Upsilon-le H* **by** *simp*
  also have ... $\le 29 + 205 * H * (nlength\ (idx + 2 * t) + nlength\ H)^2 +$
    $4 * H * (4 + nlength\ (idx + 2 * numiter) + nlength\ H) + 4 * nlength\ (Suc\ (Suc\ (idx + 2 * t))) +$
    $2 * nlength\ numiter$

using *nlength-mono assms*(*1*) **by** *simp*
also have ... $\leq$ *29* + *205* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *t*) + *nlength H*)^2 +
4 $*$ *H* $*$ (*4* + *nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*) + *4* $*$ *nlength* (*idx* + *2* $*$ *numiter*) +
2 $*$ *nlength numiter*
using *nlength-mono assms*(*1*) **by** *simp*
also have ... $\leq$ *29* + *205* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2 +
4 $*$ *H* $*$ (*4* + *nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*) + *4* $*$ *nlength* (*idx* + *2* $*$ *numiter*) +
2 $*$ *nlength numiter*
using *nlength-mono assms*(*1*) **by** *simp*
also have ... $\leq$ *29* + *205* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2 +
4 $*$ *H* $*$ (*4* + *nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*) + *6* $*$ *nlength* (*idx* + *2* $*$ *numiter*)
using *nlength-mono* **by** *simp*
also have ... $\leq$ *29* + *205* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2 +
4 $*$ *H* $*$ (*4* + *nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*) + *6* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)
using *nlength-mono* **by** *simp*
also have ... = *29* + *205* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2 +
16 $*$ *H* + *4* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*) + *6* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) +
nlength H)
by *algebra*
also have ... $\leq$ *29* + *205* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2 +
16 $*$ *H* + *4* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*) + *2* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) +
nlength H)
proof −
have *6* $\leq$ *2* $*$ *H*
using *H* **by** *simp*
then show *?thesis*
using *mult-le-mono1 nat-add-left-cancel-le* **by** *presburger*
qed
also have ... = *29* + *205* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2 +
16 $*$ *H* + *6* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)
by *simp*
also have ... $\leq$ *29* + *205* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2 +
16 $*$ *H* + *6* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^*2*
using *linear-le-pow* **by** *simp*
also have ... = *29* + *211* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2 + *16* $*$ *H*
by *simp*
also have ... $\leq$ *29* + *227* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2
using *H nlength-0* **by** (*simp add*: *Suc-leI*)
also have ... $\leq$ *256* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2
using *H nlength-0* **by** (*simp add*: *Suc-leI*)
finally have *?ttt* $\leq$ *ttt*
using *assms*(*2*) **by** *simp*
then have *transforms tmL5* (*tpsL t*) *ttt* (*tpsL5 t*)
using *assms*(*1*) *tmL5 transforms-monotone* **by** *blast*
then show *?thesis*
using *tpsL5-eq-tpsL* **by** *simp*
qed

lemma *tmL*:
assumes *ttt* = *numiter* $*$ (*257* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2) + *1*
shows *transforms tmL* (*tpsL 0*) *ttt* (*tpsL numiter*)
unfolding *tmL-def*
proof (*tform*)
show $\bigwedge$*i*. *i* < *numiter* $\Longrightarrow$ *read* (*tpsL i*) ! (*j* + *6*) $\neq$ $\Box$
using *jk tpsL-def read-ncontents-eq-0* **by** *simp*
show $\neg$ *read* (*tpsL numiter*) ! (*j* + *6*) $\neq$ $\Box$
using *jk tpsL-def read-ncontents-eq-0* **by** *simp*
show *numiter* $*$ (*256* $*$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2 + *2*) + *1* $\leq$ *ttt*
proof −
have *1* $\leq$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2
using *nlength-H* **by** *simp*
then have *2* $\leq$ *H* $*$ (*nlength* (*idx* + *2* $*$ *numiter*) + *nlength H*)^2
using *H* **by** (*metis add-leE mult-2 mult-le-mono nat-1-add-1 numeral-Bit1 numerals*(*1*))

then show *?thesis*
using *assms* **by** *simp*
qed
qed

lemma *tmL′*:
assumes $ttt = numiter * 257 * H * (nlength\ (idx + 2 * numiter) + nlength\ H)^2 + 1$
shows *transforms tmL tps0 ttt* (*tpsL numiter*)
using *assms tmL tpsL-eq-tps0* **by** (*simp add*: *Groups.mult-ac*(*1*))

end

end

lemma *transforms-tm-PHI7I*:
fixes *tps tps′* :: *tape list* **and** *ttt numiter H k idx* :: *nat* **and** *nss* :: *nat list list* **and** *j* :: *tapeidx*
assumes *length tps* = *k* **and** $1 < j$ **and** $j + 6 < k$
and $H \geq 3$
assumes
$tps\ !\ 1 = nlltape\ nss$
$tps\ !\ j = (\lfloor idx \rfloor_N,\ 1)$
$tps\ !\ (j + 1) = (\lfloor H \rfloor_N,\ 1)$
$tps\ !\ (j + 2) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 3) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 4) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 5) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 6) = (\lfloor numiter \rfloor_N,\ 1)$
assumes $ttt = numiter * 257 * H * (nlength\ (idx + 2 * numiter) + nlength\ H)^2 + 1$
assumes $tps' = tps$
$[j := (\lfloor idx + 2 * numiter \rfloor_N,\ 1),$
$j + 6 := (\lfloor 0 \rfloor_N,\ 1),$
$1 := nlltape\ (nss\ @\ concat\ (map\ (\lambda t.\ nll\text{-}Upsilon\ (idx + 2 * t)\ H)\ [0..{<}numiter]))]$
shows *transforms* (*tm-PHI7 j*) *tps ttt tps′*
proof −
interpret *loc*: *turing-machine-tm-PHI7 j* **.**
show *?thesis*
using *assms loc.tmL′ loc.tpsL-def loc.tmL-eq-tm-PHI7* **by** *simp*
qed

7.6.7 A Turing machine for Φ_8

The next TM expects a number *idx* on tape *j* and a number *H* on tape $j + 1$. It appends to the formula on tape 1 the formula $\Psi([idx \cdot H, (idx + 1)H), 3)$.

definition *tm-PHI8* :: *tapeidx* ⇒ *machine* **where**
tm-PHI8 j ≡
tm-setn (*j* + *2*) *3* ;;
tm-Psigamma j ;;
tm-extendl 1 (*j* + *6*)

lemma *tm-PHI8-tm*:
assumes $0 < j$ **and** $j + 7 < k$ **and** $G \geq 6$
shows *turing-machine k G* (*tm-PHI8 j*)
unfolding *tm-PHI8-def* **using** *assms tm-Psigamma-tm tm-setn-tm tm-extendl-tm* **by** *simp*

locale *turing-machine-PHI8* =
fixes *j* :: *tapeidx*
begin

definition *tm1* ≡ *tm-setn* (*j* + *2*) *3*
definition *tm2* ≡ *tm1* ;; *tm-Psigamma j*
definition *tm3* ≡ *tm2* ;; *tm-extendl 1* (*j* + *6*)

lemma *tm3-eq-tm-PHI8*: *tm3* = *tm-PHI8 j*

using *tm3-def tm2-def tm1-def tm-PHI8-def* **by** *simp*

context
fixes *tps0* :: *tape list* **and** *k idx H* :: *nat* **and** *nss* :: *nat list list*
assumes *jk*: *length tps0* = *k* *1* < *j* *j* + *7* < *k*
and *H*: $H \geq 3$
assumes *tps0*:
tps0 ! *1* = *nlltape nss*
tps0 ! *j* = ($\lfloor idx \rfloor_N$, *1*)
tps0 ! (*j* + *1*) = ($\lfloor H \rfloor_N$, *1*)
tps0 ! (*j* + *2*) = ($\lfloor [] \rfloor$, *1*)
tps0 ! (*j* + *3*) = ($\lfloor [] \rfloor$, *1*)
tps0 ! (*j* + *4*) = ($\lfloor [] \rfloor$, *1*)
tps0 ! (*j* + *5*) = ($\lfloor [] \rfloor$, *1*)
tps0 ! (*j* + *6*) = ($\lfloor [] \rfloor$, *1*)
tps0 ! (*j* + *7*) = ($\lfloor [] \rfloor$, *1*)
begin

definition *tps1* ≡ *tps0*
[*j* + *2* := ($\lfloor 3 \rfloor_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
assumes *ttt* = *14*
shows *transforms tm1 tps0 ttt tps1*
unfolding *tm1-def*
proof (*tform tps*: *tps0 tps1-def jk*)
show *tps0* ! (*j* + *2*) = ($\lfloor 0 \rfloor_N$, *1*)
using *tps0 jk canrepr-0* **by** *simp*
show *ttt* = *10* + *2* ∗ *nlength 0* + *2* ∗ *nlength 3*
using *assms nlength-3* **by** *simp*
qed

definition *tps2* ≡ *tps0*
[*j* + *2* := ($\lfloor 3 \rfloor_N$, *1*),
j + *6* := ($\lfloor$*nll-Psi* (*idx* ∗ *H*) *H 3*$\rfloor_{NLL}$, *1*)]

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *14* + *1851* ∗ *H* ^ *4* ∗ *nlength* (*Suc idx*) ^ *2*
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def* **by** (*tform tps*: *tps0 H tps1-def tps2-def jk time*: *assms canrepr-1*)

definition *tps3* ≡ *tps0*
[*1* := *nlltape* (*nss* @ *nll-Psi* (*idx* ∗ *H*) *H 3*),
j + *2* := ($\lfloor 3 \rfloor_N$, *1*),
j + *6* := ($\lfloor$*nll-Psi* (*idx* ∗ *H*) *H 3*$\rfloor_{NLL}$, *1*)]

lemma *tm3*:
assumes *ttt* = *18* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2 +
2 ∗ *nlllength* (*nll-Psi* (*idx* ∗ *H*) *H 3*)
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def* **by** (*tform tps*: *tps0 H tps2-def tps3-def jk time*: *assms*)

lemma *tm3'*:
assumes *ttt* = *18* + *1861* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2
shows *transforms tm3 tps0 ttt tps3*
proof −
have ∗: (*nlength* (*Suc idx*))^2 ≥ *1*
using *nlength-0* **by** (*simp add*: *Suc-leI*)
let *?ttt* = *18* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2 + *2* ∗ *nlllength* (*nll-Psi* (*idx* ∗ *H*) *H 3*)
have *?ttt* ≤ *18* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2 + *2* ∗ *H* ∗ (*3* + *nlength* (*idx* ∗ *H* + *H*))
using *nlllength-nll-Psi-le* **by** *simp*
also have ... = *18* + *1851* ∗ *H* ^ *4* ∗ (*nlength* (*Suc idx*))^2 + *2* ∗ *H* ∗ (*3* + *nlength* (*Suc idx* ∗ *H*))
by (*simp add*: *add.commute*)

also have ... ≤ *18 + 1851 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} *+ 2 ∗ H ∗ (3 + nlength (Suc idx) + nlength H)*
using *nlength-prod* **by** *(metis (mono-tags, lifting) ab-semigroup-add-class.add-ac(1) add-left-mono mult-le-cancel1)*
also have ... = *18 + 1851 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} *+ 6 ∗ H + 2 ∗ H ∗ nlength (Suc idx) + 2 ∗ H ∗ nlength H*
by *algebra*
also have ... ≤ *18 + 1851 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} *+ 6 ∗ H + 2 ∗ H ∗ nlength (Suc idx) + 2 ∗ H ∗ H*
using *nlength-le-n* **by** *simp*
also have ... ≤ *18 + 1851 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} *+ 6 ∗ H ^ 4 + 2 ∗ H ∗ nlength (Suc idx) + 2 ∗ H ∗ H*
using *linear-le-pow[of 4 H]* **by** *simp*
also have ... ≤ *18 + 1851 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} *+ 6 ∗ H ^ 4 + 2 ∗ H ^ 4 ∗ nlength (Suc idx) + 2 ∗ H ∗ H*
using *linear-le-pow[of 4 H]* **by** *simp*
also have ... ≤ *18 + 1857 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} *+ 2 ∗ H ^ 4 ∗ nlength (Suc idx) + 2 ∗ H ∗ H*
using ∗ **by** *simp*
also have ... ≤ *18 + 1859 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} *+ 2 ∗ H ∗ H*
using *linear-le-pow[of 2 nlength (Suc idx)]* **by** *simp*
also have ... = *18 + 1859 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} *+ 2 ∗ H ^ 2*
by *algebra*
also have ... ≤ *18 + 1859 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} *+ 2 ∗ H ^ 4*
using *pow-mono′[of 2 4 H]* **by** *simp*
also have ... ≤ *18 + 1861 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2}
using ∗ **by** *simp*
finally have *?ttt* ≤ *18 + 1861 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2} **.**
then show *?thesis*
using *tm3 assms transforms-monotone* **by** *simp*
qed

end

end

lemma *transforms-tm-PHI8I*:
fixes *j :: tapeidx*
fixes *tps tps′ :: tape list* **and** *ttt k idx H :: nat* **and** *nss :: nat list list*
assumes *length tps = k* **and** *1 < j* **and** *j + 7 < k*
and *H ≥ 3*
assumes
tps ! 1 = nlltape nss
tps ! j = (⌊idx⌋$_N$, 1)
tps ! (j + 1) = (⌊H⌋$_N$, 1)
tps ! (j + 2) = (⌊[]⌋, 1)
tps ! (j + 3) = (⌊[]⌋, 1)
tps ! (j + 4) = (⌊[]⌋, 1)
tps ! (j + 5) = (⌊[]⌋, 1)
tps ! (j + 6) = (⌊[]⌋, 1)
tps ! (j + 7) = (⌊[]⌋, 1)
assumes *tps′ = tps*
[1 := nlltape (nss @ nll-Psi (idx ∗ H) H 3),
j + 2 := (⌊3⌋$_N$, 1),
j + 6 := (⌊nll-Psi (idx ∗ H) H 3⌋$_{NLL}$, 1)]
assumes *ttt = 18 + 1861 ∗ H ^ 4 ∗ (nlength (Suc idx))*^{2}
shows *transforms (tm-PHI8 j) tps ttt tps′*
proof −
interpret *loc*: *turing-machine-PHI8 j* **.**
show *?thesis*
using *loc.tps3-def loc.tm3′ loc.tm3-eq-tm-PHI8 assms* **by** *metis*
qed

7.6.8 A Turing machine for Φ_9

The CNF formula $\Phi_9 = \bigwedge_{t=1}^{T'}$ is the most complicated part of Φ. Clearly, the main task here is to generate the formulas χ_t

A Turing machine for χ_t

A lemma that will help with some time bounds:

lemma *pow2-le-2pow2*: $z \,\hat{}\, 2 \le 2 \,\hat{}\, (2{*}z)$ **for** z :: *nat*
proof (*induction z*)
  case *0*
  then show *?case*
    by *simp*
next
  case (*Suc z*)
  show *?case*
  proof (*cases z = 0*)
    case *True*
    then show *?thesis*
      by *simp*
  next
    case *False*
    have *Suc z ^ 2 = z ^ 2 + 2 ∗ z + 1*
    by (*metis Suc-eq-plus1 ab-semigroup-add-class.add-ac(1) nat-mult-1-right one-power2 plus-1-eq-Suc power2-sum*)
    also have *... ≤ z ^ 2 + 3 ∗ z*
      using *False* **by** *simp*
    also have *... ≤ z ^ 2 + 3 ∗ z ^ 2*
      by (*simp add: linear-le-pow*)
    also have *... = 2^2 ∗ z ^ 2*
      by *simp*
    also have *... ≤ 2^2 ∗ 2 ^ (2 ∗ z)*
      using *Suc* **by** *simp*
    also have *... = 2 ^ (2 ∗ Suc z)*
      by (*simp add: power-add*)
    finally show *?thesis* **.**
  qed
qed

The next Turing machine can be used to generate χ_t. It expects on tape 1 a CNF formula, on tape j_1 the list of positions of M's input tape head, on tape j_2 the list of positions of M's output tape head, on tapes $j_3, \dots, j_3+3$ the numbers N, G, Z, and T, on tape j_3+4 the formula ψ, on tape j_3+5 the formula ψ', and finally on tape j_3+6 the number t. The TM appends the formula χ_t to the formula on tape 1, which should be thought of as an unfinished version of Φ.

The TM first computes *prev*(t) using *tm-prev* and compares it with t. Depending on the outcome of this comparison it generates either ϱ_t or ϱ'_t by concatenating ranges of numbers generated using *tm-range*. Then the TM uses *tm-relabel* to compute $\varrho_t(\psi)$ or $\varrho'_t(\psi')$. The result equals χ_t and is appended to tape 1. Finally t is incremented and T is decremented. This is so the TM can be used inside a while loop that initializes T with T' and t with 1.

definition *tm-chi* :: *tapeidx ⇒ tapeidx ⇒ tapeidx ⇒ machine* **where**
  tm-chi j1 j2 j3 ≡
    tm-prev j2 (j3 + 6) ;;
    tm-equalsn (j3 + 11) (j3 + 6) (j3 + 13) ;;
    tm-decr (j3 + 6) ;;
    tm-mult (j3 + 6) (j3 + 2) (j3 + 7) ;;
    tm-add j3 (j3 + 7) ;;
    tm-range (j3 + 7) (j3 + 2) (j3 + 8) ;;
    tm-extend-erase (j3 + 12) (j3 + 8) ;;
    tm-setn (j3 + 7) 0 ;;
    IF λrs. rs ! (j3 + 13) = □ THEN
      tm-mult (j3 + 11) (j3 + 2) (j3 + 7) ;;
      tm-add j3 (j3 + 7) ;;
      tm-range (j3 + 7) (j3 + 2) (j3 + 8) ;;
      tm-extend-erase (j3 + 12) (j3 + 8) ;;
      tm-setn (j3 + 7) 0
    ELSE
      []

```
    ENDIF ;;
    tm-incr (j3 + 6) ;;
    tm-mult (j3 + 6) (j3 + 2) (j3 + 7) ;;
    tm-add j3 (j3 + 7) ;;
    tm-range (j3 + 7) (j3 + 2) (j3 + 8) ;;
    tm-extend-erase (j3 + 12) (j3 + 8) ;;
    tm-setn (j3 + 11) 0 ;;
    tm-nth j1 (j3 + 6) (j3 + 11) 4 ;;
    tm-setn (j3 + 7) 0 ;;
    tm-mult (j3 + 11) (j3 + 1) (j3 + 7) ;;
    tm-range (j3 + 7) (j3 + 1) (j3 + 8) ;;
    tm-extend-erase (j3 + 12) (j3 + 8) ;;
    tm-setn (j3 + 7) 0 ;;
    tm-erase-cr (j3 + 11) ;;
    tm-cr (j3 + 12) ;;
    IF λrs. rs ! (j3 + 13) = □ THEN
      tm-relabel (j3 + 4) (j3 + 11)
    ELSE
      tm-erase-cr (j3 + 13) ;;
      tm-relabel (j3 + 5) (j3 + 11)
    ENDIF ;;
    tm-erase-cr (j3 + 12) ;;
    tm-extendl-erase 1 (j3 + 11) ;;
    tm-incr (j3 + 6) ;;
    tm-decr (j3 + 3)

lemma tm-chi-tm:
  assumes 0 < j1 and j1 < j2 and j2 < j3 and j3 + 17 < k and G ≥ 6
  shows turing-machine k G (tm-chi j1 j2 j3)
  unfolding tm-chi-def
  using tm-prev-tm tm-equalsn-tm tm-decr-tm tm-mult-tm tm-add-tm tm-range-tm tm-extend-erase-tm
    tm-setn-tm tm-incr-tm tm-nth-tm tm-erase-cr-tm tm-relabel-tm Nil-tm tm-cr-tm tm-extendl-erase-tm
    assms turing-machine-branch-turing-machine
  by simp

locale turing-machine-chi =
  fixes j1 j2 j3 :: tapeidx
begin

definition tm1 ≡ tm-prev j2 (j3 + 6)
definition tm2 ≡ tm1 ;; tm-equalsn (j3 + 11) (j3 + 6) (j3 + 13)

definition tm3 ≡ tm2 ;; tm-decr (j3 + 6)
definition tm4 ≡ tm3 ;; tm-mult (j3 + 6) (j3 + 2) (j3 + 7)
definition tm5 ≡ tm4 ;; tm-add j3 (j3 + 7)
definition tm6 ≡ tm5 ;; tm-range (j3 + 7) (j3 + 2) (j3 + 8)
definition tm7 ≡ tm6 ;; tm-extend-erase (j3 + 12) (j3 + 8)
definition tm8 ≡ tm7 ;; tm-setn (j3 + 7) 0

definition tmT1 ≡ tm-mult (j3 + 11) (j3 + 2) (j3 + 7)
definition tmT2 ≡ tmT1 ;; tm-add j3 (j3 + 7)
definition tmT3 ≡ tmT2 ;; tm-range (j3 + 7) (j3 + 2) (j3 + 8)
definition tmT4 ≡ tmT3 ;; tm-extend-erase (j3 + 12) (j3 + 8)
definition tmT5 ≡ tmT4 ;; tm-setn (j3 + 7) 0

definition tm89 ≡ IF λrs. rs ! (j3 + 13) = □ THEN tmT5 ELSE [] ENDIF
definition tm10 ≡ tm8 ;; tm89

definition tm11 ≡ tm10 ;; tm-incr (j3 + 6)
definition tm13 ≡ tm11 ;; tm-mult (j3 + 6) (j3 + 2) (j3 + 7)
definition tm14 ≡ tm13 ;; tm-add j3 (j3 + 7)
definition tm15 ≡ tm14 ;; tm-range (j3 + 7) (j3 + 2) (j3 + 8)
definition tm16 ≡ tm15 ;; tm-extend-erase (j3 + 12) (j3 + 8)
```

definition *tm17* ≡ *tm16* ;; *tm-setn* (*j3* + *11*) *0*
definition *tm18* ≡ *tm17* ;; *tm-nth j1* (*j3* + *6*) (*j3* + *11*) *4*
definition *tm19* ≡ *tm18* ;; *tm-setn* (*j3* + *7*) *0*
definition *tm20* ≡ *tm19* ;; *tm-mult* (*j3* + *11*) (*j3* + *1*) (*j3* + *7*)
definition *tm21* ≡ *tm20* ;; *tm-range* (*j3* + *7*) (*j3* + *1*) (*j3* + *8*)
definition *tm22* ≡ *tm21* ;; *tm-extend-erase* (*j3* + *12*) (*j3* + *8*)
definition *tm23* ≡ *tm22* ;; *tm-setn* (*j3* + *7*) *0*
definition *tm24* ≡ *tm23* ;; *tm-erase-cr* (*j3* + *11*)
definition *tm25* ≡ *tm24* ;; *tm-cr* (*j3* + *12*)

definition *tmE1* ≡ *tm-erase-cr* (*j3* + *13*)
definition *tmE2* ≡ *tmE1* ;; *tm-relabel* (*j3* + *5*) (*j3* + *11*)
definition *tmTT1* ≡ *tm-relabel* (*j3* + *4*) (*j3* + *11*)

definition *tm2526* ≡ *IF* λ*rs. rs* ! (*j3* + *13*) = □ *THEN tmTT1 ELSE tmE2 ENDIF*
definition *tm26* ≡ *tm25* ;; *tm2526*
definition *tm27* ≡ *tm26* ;; *tm-erase-cr* (*j3* + *12*)
definition *tm28* ≡ *tm27* ;; *tm-extendl-erase 1* (*j3* + *11*)
definition *tm29* ≡ *tm28* ;; *tm-incr* (*j3* + *6*)
definition *tm30* ≡ *tm29* ;; *tm-decr* (*j3* + *3*)

lemma *tm30-eq-tm-chi*: *tm30* = *tm-chi j1 j2 j3*
unfolding *tm30-def tm29-def tm28-def tm27-def tm26-def tm25-def tm24-def tm23-def tm22-def tm21-def tm20-def*
tm19-def tm18-def tm17-def tm16-def tm15-def tm14-def tm13-def tm11-def tm10-def
tm8-def tm7-def tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def
tm89-def tmE1-def tmE2-def tmTT1-def tm2526-def
tmT5-def tmT4-def tmT3-def tmT2-def tmT1-def
tm-chi-def
by *simp*

context
fixes *tps0* :: *tape list* **and** *k G N Z T′ T t* :: *nat* **and** *hp0 hp1* :: *nat list* **and** ψ ψ' :: *formula*
fixes *nss* :: *nat list list*
assumes *jk*: *length tps0* = *k 1* < *j1 j1* < *j2 j2* < *j3 j3* + *17* < *k*
and *G*: *G* ≥ *3*
and *Z*: *Z* = *3* ∗ *G*
and *N*: *N* ≥ *1*
and *len-hp0*: *length hp0* = *Suc T′*
and *hp0*: ∀ *i*<*length hp0*. *hp0* ! *i* ≤ *T′*
and *len-hp1*: *length hp1* = *Suc T′*
and *hp1*: ∀ *i*<*length hp1*. *hp1* ! *i* ≤ *T′*
and *t*: *0* < *t t* ≤ *T′*
and *T*: *0* < *T T* ≤ *T′*
and *T′*: *T′* < *N*
and *psi*: *variables* ψ ⊆ {..<*3*∗*Z*+*G*} *fsize* ψ ≤ (*3*∗*Z*+*G*) ∗ *2* ^ (*3*∗*Z*+*G*) *length* ψ ≤ *2* ^ (*3*∗*Z*+*G*)
and *psi′*: *variables* ψ' ⊆ {..<*2*∗*Z*+*G*} *fsize* ψ' ≤ (*2*∗*Z*+*G*) ∗ *2* ^ (*2*∗*Z*+*G*) *length* ψ' ≤ *2* ^ (*2*∗*Z*+*G*)
assumes *tps0*:
tps0 ! *1* = *nlltape nss*
tps0 ! *j1* = ($\lfloor$*hp0*$\rfloor_{NL}$, *1*)
tps0 ! *j2* = ($\lfloor$*hp1*$\rfloor_{NL}$, *1*)
tps0 ! *j3* = ($\lfloor$*N*$\rfloor_N$, *1*)
tps0 ! (*j3* + *1*) = ($\lfloor$*G*$\rfloor_N$, *1*)
tps0 ! (*j3* + *2*) = ($\lfloor$*Z*$\rfloor_N$, *1*)
tps0 ! (*j3* + *3*) = ($\lfloor$*T*$\rfloor_N$, *1*)
tps0 ! (*j3* + *4*) = ($\lfloor$*formula-n* $\psi$$\rfloor_{NLL}$, *1*)
tps0 ! (*j3* + *5*) = ($\lfloor$*formula-n* $\psi'$$\rfloor_{NLL}$, *1*)
tps0 ! (*j3* + *6*) = ($\lfloor$*t*$\rfloor_N$, *1*)
$\bigwedge$*i*. *6* < *i* ⟹ *i* < *17* ⟹ *tps0* ! (*j3* + *i*) = ($\lfloor$[]$\rfloor$, *1*)
begin

lemma *Z-ge-1*: *Z* ≥ *1*
using *G Z* **by** *simp*

lemma *Z-ge-9*: *Z* ≥ *9*
  using *G Z* **by** *simp*

lemma *T′-ge-1*: *T′* ≥ *1*
  using *t* **by** *simp*

lemma *tps0′*: $\bigwedge$*i*. *1* ≤ *i* ⟹ *i* < *11* ⟹ *tps0* ! (*j3* + *6* + *i*) = (⌊[]⌋, *1*)
proof −
  fix *i* :: *nat*
  assume *i*: *1* ≤ *i i* < *11*
  then have *6* < *6* + *i 6* + *i* < *17*
    by *simp-all*
  then have *tps0* ! (*j3* + (*6* + *i*)) = (⌊[]⌋, *1*)
    using *tps0*(*11*) **by** *simp*
  then show *tps0* ! (*j3* + *6* + *i*) = (⌊[]⌋, *1*)
    by (*metis group-cancel.add1*)
qed

The simplifier turns $j3 + 6 + 3$ into $9 + j3$. The next lemma helps with that.

lemma *tps0-sym*: $\bigwedge$*i*. *6* < *i* ⟹ *i* < *17* ⟹ *tps0* ! (*i* + *j3*) = (⌊[]⌋, *1*)
  using *tps0*(*11*) **by** (*simp add*: *add.commute*)

lemma *previous-hp1-le*: *previous hp1 t* ≤ *t*
  using *len-hp1 hp1 t*(*2*) *previous-le*[*of hp1 t*] **by** *simp*

definition *tps1* ≡ *tps0*
  [*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*)]

lemma *tm1* [*transforms-intros*]:
  assumes *ttt* = *71* + *153* ∗ *nllength hp1* ^ *3*
  shows *transforms tm1 tps0 ttt tps1*
  unfolding *tm1-def*
proof (*tform tps*: *canrepr-0 tps0-sym tps0 tps1-def jk t len-hp1 time*: *assms*)
  show *tps1* = *tps0*[*j3* + *6* + *5* := (⌊*previous hp1 t*⌋$_N$, *1*)]
    using *tps1-def* **by** (*simp add*: *add.commute*)
qed

definition *tps2* ≡ *tps0*
  [*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*)]

lemma *tm2* [*transforms-intros*]:
  assumes *ttt* = *78* + *153* ∗ *nllength hp1* ^ *3* + *3* ∗ *nlength* (*min* (*previous hp1 t*) *t*)
  shows *transforms tm2 tps0 ttt tps2*
  unfolding *tm2-def*
proof (*tform tps*: *tps0 tps1-def tps2-def jk time*: *assms*)
  show *tps1* ! (*j3* + *13*) = (⌊*0*⌋$_N$, *1*)
    using *tps1-def tps0*(*11*) *canrepr-0* **by** *simp*
  show (*0*::*nat*) ≤ *1*
    by *simp*
qed

definition *tps3* ≡ *tps0*
  [*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
   j3 + *6* := (⌊*t* − *1*⌋$_N$, *1*)]

lemma *tm3* [*transforms-intros*]:
  assumes *ttt* = *86* + *153* ∗ *nllength hp1* ^ *3* + *3* ∗ *nlength* (*min* (*previous hp1 t*) *t*) + *2* ∗ *nlength t*
  shows *transforms tm3 tps0 ttt tps3*
  unfolding *tm3-def* **by** (*tform tps*: *tps0 tps2-def tps3-def jk assms*)

definition *tps4* ≡ *tps0*
  [*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
   j3 + *6* := (⌊*t* − *1*⌋$_N$, *1*),
   j3 + *7* := (⌊(*t* − *1*) ∗ *Z*⌋$_N$, *1*)]

lemma *tm4* [*transforms-intros*]:
  assumes *ttt* = *90* + *153* ∗ *nllength hp1* ^ *3* + *3* ∗ *nlength* (*min* (*previous hp1 t*) *t*) + *2* ∗ *nlength t* +
    26 ∗ (*nlength* (*t* − *1*) + *nlength Z*) ^ *2*
  shows *transforms tm4 tps0 ttt tps4*
  unfolding *tm4-def*
proof (*tform tps*: *tps0 tps3-def tps4-def jk*)
  show *tps3* ! (*j3* + *7*) = (⌊*0*⌋$_N$, *1*)
    using *tps3-def jk canrepr-0 tps0* **by** *simp*
  show *ttt* = *86* + *153* ∗ *nllength hp1* ^ *3* + *3* ∗ *nlength* (*min* (*previous hp1 t*) *t*) +
    2 ∗ *nlength t* + (*4* + *26* ∗ (*nlength* (*t* − *Suc 0*) + *nlength Z*) ∗ (*nlength* (*t* − *Suc 0*) + *nlength Z*))
  proof −
    have (*26* ∗ *nlength* (*t* − *Suc 0*) + *26* ∗ *nlength Z*) ∗ (*nlength* (*t* − *Suc 0*) + *nlength Z*) =
        26 ∗ (*nlength* (*t* − *Suc 0*) + *nlength Z*) ^ *2*
      by *algebra*
    then show *?thesis*
      using *assms* **by** *simp*
  qed
qed

definition *tps5* ≡ *tps0*
  [*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
   j3 + *6* := (⌊*t* − *1*⌋$_N$, *1*),
   j3 + *7* := (⌊*N* + (*t* − *1*) ∗ *Z*⌋$_N$, *1*)]

lemma *tm5* [*transforms-intros*]:
  assumes *ttt* = *100* + *153* ∗ *nllength hp1* ^ *3* + *3* ∗ *nlength* (*min* (*previous hp1 t*) *t*) + *2* ∗ *nlength t* +
    26 ∗ (*nlength* (*t* − *1*) + *nlength Z*) ^ *2* + *3* ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*))
  shows *transforms tm5 tps0 ttt tps5*
  unfolding *tm5-def* **by** (*tform tps*: *tps0 tps4-def tps5-def jk time*: *assms*)

definition *tps6* ≡ *tps0*
  [*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
   j3 + *6* := (⌊*t* − *1*⌋$_N$, *1*),
   j3 + *7* := (⌊*N* + (*t* − *1*) ∗ *Z*⌋$_N$, *1*),
   j3 + *8* := (⌊[*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*]⌋$_{NL}$, *1*)]

lemma *tm6* [*transforms-intros*]:
  assumes *ttt* = *100* + *153* ∗ *nllength hp1* ^ *3* + *3* ∗ *nlength* (*min* (*previous hp1 t*) *t*) + *2* ∗ *nlength t* +
    26 ∗ (*nlength* (*t* − *1*) + *nlength Z*) ^ *2* + *3* ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*)) +
    Suc Z ∗ (*43* + *9* ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*))
  shows *transforms tm6 tps0 ttt tps6*
  unfolding *tm6-def*
  by (*tform tps*: *nlcontents-Nil canrepr-0 tps0-sym tps0 tps5-def tps6-def jk time*: *assms*)

definition *tps7* ≡ *tps0*
  [*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
   j3 + *6* := (⌊*t* − *1*⌋$_N$, *1*),
   j3 + *7* := (⌊*N* + (*t* − *1*) ∗ *Z*⌋$_N$, *1*),
   j3 + *12* := *nltape* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*]]

lemma *tm7* [*transforms-intros*]:
  assumes *ttt* = *111* + *153* ∗ *nllength hp1* ^ *3* + *3* ∗ *nlength* (*min* (*previous hp1 t*) *t*) + *2* ∗ *nlength t* +
    26 ∗ (*nlength* (*t* − *1*) + *nlength Z*) ^ *2* + *3* ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*)) +
    Suc Z ∗ (*43* + *9* ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) +

4 ∗ nllength [N + (t − Suc 0) ∗ Z..<N + (t − Suc 0) ∗ Z + Z]
shows *transforms tm7 tps0 ttt tps7*
unfolding *tm7-def*
proof (*tform tps*: *tps0 tps6-def tps7-def jk time*: *assms*)
show *tps6* ! (*j3* + *12*) = *nltape* []
using *tps6-def jk nlcontents-Nil tps0* **by** *force*
show *tps7* = *tps6*
[*j3* + *12* := *nltape* ([] @ [*N* + (*t* − *Suc 0*) ∗ *Z*..<*N* + (*t* − *Suc 0*) ∗ *Z* + *Z*]),
j3 + *8* := (⌊[]⌋, *1*)]
unfolding *tps7-def tps6-def*
using *tps0 jk list-update-id*[*of tps0 j3* + *8*]
by (*simp add*: *list-update-swap*)
qed

definition *tps8* ≡ *tps0*
[*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*),
j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
j3 + *6* := (⌊*t* − *1*⌋$_N$, *1*),
j3 + *7* := (⌊*0*⌋$_N$, *1*),
j3 + *12* := *nltape* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*]]

lemma *tm8*:
assumes *ttt* = *121* + *153* ∗ *nllength hp1* ^ *3* + *3* ∗ *nlength* (*min* (*previous hp1 t*) *t*) +
2 ∗ *nlength t* + *26* ∗ (*nlength* (*t* − *1*) + *nlength Z*) ^ *2* + *3* ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*)) +
Suc Z ∗ (*43* + *9* ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) + *4* ∗ *nllength* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z*
+ *Z*] +
2 ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z*)
shows *transforms tm8 tps0 ttt tps8*
unfolding *tm8-def* **by** (*tform tps*: *tps0 tps7-def tps8-def jk time*: *assms*)

For the next upper bound we have no scruples replacing $\log T'$, $\log N$, and $\log Z$ by T', N and Z, respectively. All values are polynomial in n (Z is even a constant), so the overall polynomiality is not in jeopardy.

lemma *nllength-le*:
fixes *nmax* :: *nat* **and** *ns* :: *nat list*
assumes ∀ *n*∈*set ns*. *n* ≤ *nmax*
shows *nllength ns* ≤ *Suc nmax* ∗ *length ns*
proof −
have *nllength ns* ≤ *Suc* (*nlength nmax*) ∗ *length ns*
using *assms nllength-le-len-mult-max* **by** *simp*
then show *?thesis*
using *nlength-le-n* **by** (*meson Suc-le-mono dual-order.trans mult-le-mono1*)
qed

lemma *nllength-upt-le*:
fixes *a b* :: *nat*
shows *nllength* [*a*..<*b*] ≤ *Suc b* ∗ (*b* − *a*)
proof −
have *nllength* [*a*..<*b*] ≤ *Suc* (*nlength b*) ∗ (*b* − *a*)
using *nllength-upt-le-len-mult-max* **by** *simp*
then show *?thesis*
using *nlength-le-n* **by** (*meson Suc-le-mono dual-order.trans mult-le-mono1*)
qed

lemma *nllength-hp1*: *nllength hp1* ≤ *Suc T′* ∗ *Suc T′*
proof −
have ∀ *n*∈*set hp1*. *n* ≤ *T′*
using *hp1* **by** (*metis in-set-conv-nth*)
then have *nllength hp1* ≤ *Suc T′* ∗ *Suc T′*
using *nllength-le*[*of hp1 T′*] *len-hp1* **by** *simp*
then show *?thesis*
by *simp*
qed

definition *ttt8* ≡ *168* + *153* ∗ *Suc T′* ^ *6* + *5* ∗ *t* + *26* ∗ (*t* + *Z*) ^ *2* + *47* ∗ *Z* + *15* ∗ *Z* ∗ (*N* + *t* ∗ *Z*)

lemma *tm8′* [*transforms-intros*]: *transforms tm8 tps0 ttt8 tps8*
proof −
  let *?s* = *88* + *153* ∗ *nllength hp1* ^ *3* + *3* ∗ *nlength* (*min* (*previous hp1 t*) *t*) + *2* ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z*)
  let *?ttt* = *121* + *153* ∗ *nllength hp1* ^ *3* + *3* ∗ *nlength* (*min* (*previous hp1 t*) *t*) +
    2 ∗ *nlength t* + *26* ∗ (*nlength* (*t* − *1*) + *nlength Z*) ^ *2* + *3* ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*)) +
    Suc Z ∗ (*43* + *9* ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) + *4* ∗ *nllength* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*] +
    2 ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z*)
  have *?ttt* = *?s* + *33* + *2* ∗ *nlength t* + *26* ∗ (*nlength* (*t* − *1*) + *nlength Z*) ^ *2* + *3* ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*)) +
      Suc Z ∗ (*43* + *9* ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) + *4* ∗ *nllength* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*]
    by *simp*
  also have ... ≤ *?s* + *33* + *2* ∗ *t* + *26* ∗ (*nlength* (*t* − *1*) + *nlength Z*) ^ *2* +
      3 ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) +
      4 ∗ *nllength* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*]
    using *nlength-le-n* **by** *simp*
  also have ... ≤ *?s* + *33* + *2* ∗ *t* + *26* ∗ ((*t* − *1*) + *nlength Z*) ^ *2* +
      3 ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) +
      4 ∗ *nllength* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*]
    using *nlength-le-n* **by** *simp*
  also have ... ≤ *?s* + *33* + *2* ∗ *t* + *26* ∗ ((*t* − *1*) + *Z*) ^ *2* +
      3 ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ *nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) +
      4 ∗ *nllength* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*]
    using *nlength-le-n* **by** *simp*
  also have ... ≤ *?s* + *33* + *2* ∗ *t* + *26* ∗ ((*t* − *1*) + *Z*) ^ *2* +
      3 ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ (*N* + (*t* − *1*) ∗ *Z* + *Z*)) +
      4 ∗ *nllength* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*]
    using *nlength-le-n add-le-mono le-refl mult-le-mono* **by** *presburger*
  also have ... ≤ *?s* + *33* + *2* ∗ *t* + *26* ∗ ((*t* − *1*) + *Z*) ^ *2* +
      3 ∗ *max* (*nlength N*) (*nlength* ((*t* − *1*) ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ (*N* + (*t* − *1*) ∗ *Z* + *Z*)) +
      4 ∗ *Suc* (*nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) ∗ *Z*
  proof −
    have *nllength* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*] ≤ *Suc* (*nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) ∗ *Z*
      using *nllength-le-len-mult-max*[*of* [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*] *N* + (*t* − *1*) ∗ *Z* + *Z*]
      by *simp*
    then show *?thesis*
      by *linarith*
  qed
  also have ... = *?s* + *33* + *2* ∗ *t* + *26* ∗ ((*t* − *1*) + *Z*) ^ *2* +
      3 ∗ *nlength* (*max N* ((*t* − *1*) ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ (*N* + (*t* − *1*) ∗ *Z* + *Z*)) +
      4 ∗ *Suc* (*nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) ∗ *Z*
    using *max-nlength* **by** *simp*
  also have ... ≤ *?s* + *33* + *2* ∗ *t* + *26* ∗ ((*t* − *1*) + *Z*) ^ *2* +
      3 ∗ *max N* ((*t* − *1*) ∗ *Z*) + *Suc Z* ∗ (*43* + *9* ∗ (*N* + (*t* − *1*) ∗ *Z* + *Z*)) +
      4 ∗ *Suc* (*nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) ∗ *Z*
    using *nlength-le-n* **by** *simp*
  also have ... = *?s* + *33* + *2* ∗ *t* + *26* ∗ ((*t* − *1*) + *Z*) ^ *2* +
      3 ∗ *max N* ((*t* − *1*) ∗ *Z*) + *Suc Z* ∗ (*43* + *9* ∗ (*N* + *t* ∗ *Z*)) +
      4 ∗ *Suc* (*nlength* (*N* + (*t* − *1*) ∗ *Z* + *Z*)) ∗ *Z*
    using *t* **by** (*smt* (*verit*) *ab-semigroup-add-class.add-ac*(*1*) *add.commute diff-Suc-1 gr0-implies-Suc mult-Suc*)
  also have ... ≤ *?s* + *33* + *2* ∗ *t* + *26* ∗ ((*t* − *1*) + *Z*) ^ *2* +
      3 ∗ *max N* ((*t* − *1*) ∗ *Z*) + *Suc Z* ∗ (*43* + *9* ∗ (*N* + *t* ∗ *Z*)) +
      4 ∗ *Suc* (*N* + (*t* − *1*) ∗ *Z* + *Z*) ∗ *Z*
    using *nlength-le-n Suc-le-mono add-le-mono le-refl mult-le-mono* **by** *presburger*
  also have ... = *?s* + *33* + *2* ∗ *t* + *26* ∗ ((*t* − *1*) + *Z*) ^ *2* +
      3 ∗ *max N* ((*t* − *1*) ∗ *Z*) + *Suc Z* ∗ (*43* + *9* ∗ (*N* + *t* ∗ *Z*)) +
      4 ∗ *Suc* (*N* + *t* ∗ *Z*) ∗ *Z*
     by (*smt* (*verit*, *del-insts*) *Suc-eq-plus1 Suc-leI add.commute add.left-commute add-leD2 le-add-diff-inverse*

mult.commute mult-Suc-right t(1))
  also have ... $\leq$ *?s + 33 + 2 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 +*
    3 ∗ (N + ((t − 1) ∗ Z)) + Suc Z ∗ (43 + 9 ∗ (N + t ∗ Z)) + 4 ∗ Suc (N + t ∗ Z) ∗ Z
   by *simp*
  also have ... $\leq$ *?s + 33 + 2 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 +*
    3 ∗ (N + t ∗ Z) + Suc Z ∗ (43 + 9 ∗ (N + t ∗ Z)) + 4 ∗ Suc (N + t ∗ Z) ∗ Z
   by *simp*
  also have ... = *?s + 33 + 2 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 +*
    3 ∗ (N + t ∗ Z) + Suc Z ∗ (43 + 9 ∗ (N + t ∗ Z)) + (4 + 4 ∗ (N + t ∗ Z)) ∗ Z
   by *simp*
  also have ... $\leq$ *?s + 33 + 2 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 +*
    3 ∗ (N + t ∗ Z) + Suc Z ∗ (43 + 9 ∗ (N + t ∗ Z)) + (4 + 4 ∗ (N + t ∗ Z)) ∗ Suc Z
   by *simp*
  also have ... = *?s + 33 + 2 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 + 3 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))*
   by *algebra*
  also have ... = *121 + 153 ∗ nllength hp1 ^ 3 + 3 ∗ nlength (min (previous hp1 t) t) + 2 ∗ nlength (N + (t − 1) ∗ Z) +*
    2 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 + 3 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   by *simp*
  also have ... $\leq$ *121 + 153 ∗ nllength hp1 ^ 3 + 3 ∗ nlength t + 2 ∗ nlength (N + (t − 1) ∗ Z) +*
    2 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 + 3 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   using *previous-hp1-le nlength-mono* **by** *simp*
  also have ... $\leq$ *121 + 153 ∗ nllength hp1 ^ 3 + 2 ∗ nlength (N + (t − 1) ∗ Z) +*
    5 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 + 3 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   using *nlength-le-n* **by** *simp*
  also have ... $\leq$ *121 + 153 ∗ nllength hp1 ^ 3 + 2 ∗ (N + (t − 1) ∗ Z) +*
    5 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 + 3 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   using *nlength-le-n add-le-mono1 mult-le-mono2 nat-add-left-cancel-le* **by** *presburger*
  also have ... $\leq$ *121 + 153 ∗ nllength hp1 ^ 3 + 2 ∗ (N + t ∗ Z) +*
    5 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 + 3 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   by *simp*
  also have ... = *121 + 153 ∗ nllength hp1 ^ 3 +*
    5 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 + 5 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   by *simp*
  also have ... $\leq$ *121 + 153 ∗ (Suc T' ∗ Suc T') ^ 3 +*
    5 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 + 5 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   using *nllength-hp1* **by** *simp*
  also have ... = *121 + 153 ∗ (Suc T' ^ 2)^3 +*
    5 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 + 5 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   by *algebra*
  also have ... = *121 + 153 ∗ Suc T' ^ 6 +*
    5 ∗ t + 26 ∗ ((t − 1) + Z) ^ 2 + 5 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   by *simp*
  also have ... $\leq$ *121 + 153 ∗ Suc T' ^ 6 +*
    5 ∗ t + 26 ∗ (t + Z) ^ 2 + 5 ∗ (N + t ∗ Z) + Suc Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   by *simp*
  also have ... = *121 + 153 ∗ Suc T' ^ 6 +*
    5 ∗ t + 26 ∗ (t + Z) ^ 2 + 5 ∗ (N + t ∗ Z) + 47 + 13 ∗ (N + t ∗ Z) + Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   by *simp*
  also have ... = *168 + 153 ∗ Suc T' ^ 6 +*
    5 ∗ t + 26 ∗ (t + Z) ^ 2 + 18 ∗ (N + t ∗ Z) + Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   by *simp*
  also have ... $\leq$ *168 + 153 ∗ Suc T' ^ 6 +*
    5 ∗ t + 26 ∗ (t + Z) ^ 2 + 2 ∗ Z ∗ (N + t ∗ Z) + Z ∗ (47 + 13 ∗ (N + t ∗ Z))
   using *Z-ge-9* **by** (*metis add-le-mono add-le-mono1 mult-2 mult-le-mono1 nat-add-left-cancel-le numeral-Bit0*)
  also have ... = *168 + 153 ∗ Suc T' ^ 6 +*
    5 ∗ t + 26 ∗ (t + Z) ^ 2 + 2 ∗ Z ∗ (N + t ∗ Z) + 47 ∗ Z + 13 ∗ Z ∗ (N + t ∗ Z)
   by *algebra*
  also have ... = *168 + 153 ∗ Suc T' ^ 6 + 5 ∗ t + 26 ∗ (t + Z) ^ 2 + 47 ∗ Z + 15 ∗ Z ∗ (N + t ∗ Z)*
   by *simp*
  finally have *?ttt* $\leq$ *ttt8*
   using *ttt8-def* **by** *simp*

then show *?thesis*
  using *tm8 transforms-monotone* **by** *simp*
qed

definition *tpsT1* ≡ *tps0*
  $[j3 + 11 := (\lfloor previous\ hp1\ t \rfloor_N, 1),$
  $j3 + 13 := (\lfloor previous\ hp1\ t = t \rfloor_B, 1),$
  $j3 + 6 := (\lfloor t - 1 \rfloor_N, 1),$
  $j3 + 7 := (\lfloor previous\ hp1\ t * Z \rfloor_N, 1),$
  $j3 + 12 :=$ *nltape* $[N + (t - 1) * Z..<N + (t - 1) * Z + Z]]$

lemma *tmT1* [*transforms-intros*]:
  assumes $ttt = 4 + 26 * (nlength\ (previous\ hp1\ t) + nlength\ Z)$ ^ 2
  shows *transforms tmT1 tps8 ttt tpsT1*
  unfolding *tmT1-def*
proof (*tform tps*: *tps0 tps8-def tpsT1-def jk*)
  show $ttt = 4 + 26 * (nlength\ (previous\ hp1\ t) + nlength\ Z) * (nlength\ (previous\ hp1\ t) + nlength\ Z)$
    using *assms* **by** *algebra*
  show $tpsT1 = tps8[j3 + 7 := (\lfloor previous\ hp1\ t * Z \rfloor_N, 1)]$
    unfolding *tpsT1-def tps8-def* **by** (*simp add*: *list-update-swap*[*of j3+7*])
qed

definition *tpsT2* ≡ *tps0*
  $[j3 + 11 := (\lfloor previous\ hp1\ t \rfloor_N, 1),$
  $j3 + 13 := (\lfloor previous\ hp1\ t = t \rfloor_B, 1),$
  $j3 + 6 := (\lfloor t - 1 \rfloor_N, 1),$
  $j3 + 7 := (\lfloor N + previous\ hp1\ t * Z \rfloor_N, 1),$
  $j3 + 12 :=$ *nltape* $[N + (t - 1) * Z..<N + (t - 1) * Z + Z]]$

lemma *tmT2* [*transforms-intros*]:
  assumes $ttt = 14 + 26 * (nlength\ (previous\ hp1\ t) + nlength\ Z)$ ^ 2 +
    $3 * max\ (nlength\ N)\ (nlength\ (previous\ hp1\ t * Z))$
  shows *transforms tmT2 tps8 ttt tpsT2*
  unfolding *tmT2-def* **by** (*tform tps*: *tps0 tpsT1-def tpsT2-def jk assms*)

definition *tpsT3* ≡ *tps0*
  $[j3 + 11 := (\lfloor previous\ hp1\ t \rfloor_N, 1),$
  $j3 + 13 := (\lfloor previous\ hp1\ t = t \rfloor_B, 1),$
  $j3 + 6 := (\lfloor t - 1 \rfloor_N, 1),$
  $j3 + 7 := (\lfloor N + previous\ hp1\ t * Z \rfloor_N, 1),$
  $j3 + 12 :=$ *nltape* $[N + (t - 1) * Z..<N + (t - 1) * Z + Z],$
  $j3 + 8 := (\lfloor [N + previous\ hp1\ t * Z..<N + previous\ hp1\ t * Z + Z] \rfloor_{NL}, 1)]$

lemma *tmT3* [*transforms-intros*]:
  assumes $ttt = 14 + 26 * (nlength\ (previous\ hp1\ t) + nlength\ Z)$ ^ 2 +
    $3 * max\ (nlength\ N)\ (nlength\ (previous\ hp1\ t * Z))$ +
    $Suc\ Z * (43 + 9 * nlength\ (N + previous\ hp1\ t * Z + Z))$
  shows *transforms tmT3 tps8 ttt tpsT3*
  unfolding *tmT3-def*
  by (*tform tps*: *nlcontents-Nil canrepr-0 tps0-sym tps0 tpsT2-def tpsT3-def jk time*: *assms*)

definition *tpsT4* ≡ *tps0*
  $[j3 + 11 := (\lfloor previous\ hp1\ t \rfloor_N, 1),$
  $j3 + 13 := (\lfloor previous\ hp1\ t = t \rfloor_B, 1),$
  $j3 + 6 := (\lfloor t - 1 \rfloor_N, 1),$
  $j3 + 7 := (\lfloor N + previous\ hp1\ t * Z \rfloor_N, 1),$
  $j3 + 12 :=$ *nltape*
      $([N + (t - 1) * Z..<N + (t - 1) * Z + Z]$ @
       $[N + previous\ hp1\ t * Z..<N + previous\ hp1\ t * Z + Z]),$
  $j3 + 8 := (\lfloor [] \rfloor, 1)]$

lemma *tmT4* [*transforms-intros*]:
  assumes $ttt = 25 + 26 * (nlength\ (previous\ hp1\ t) + nlength\ Z)$ ^ 2 +

3 ∗ max (nlength N) (nlength (previous hp1 t ∗ Z)) +
Suc Z ∗ (43 + 9 ∗ nlength (N + previous hp1 t ∗ Z + Z)) +
4 ∗ nllength [N + previous hp1 t ∗ Z..<N + previous hp1 t ∗ Z + Z]
shows *transforms tmT4 tps8 ttt tpsT4*
unfolding *tmT4-def* **by** (*tform tps*: *tps0 tpsT3-def tpsT4-def jk time*: *assms*)

definition *tpsT5 ≡ tps0*
[*j3 + 11* := (⌊*previous hp1 t*⌋$_N$, *1*),
j3 + 13 := (⌊*previous hp1 t = t*⌋$_B$, *1*),
j3 + 6 := (⌊*t − 1*⌋$_N$, *1*),
j3 + 7 := (⌊*0*⌋$_N$, *1*),
j3 + 12 := *nltape*
([*N + (t − 1) ∗ Z..<N + (t − 1) ∗ Z + Z*] @
[*N + previous hp1 t ∗ Z..<N + previous hp1 t ∗ Z + Z*]),
j3 + 8 := (⌊[]⌋, *1*)]

lemma *tmT5* [*transforms-intros*]:
assumes *ttt = 35 + 26 ∗ (nlength (previous hp1 t) + nlength Z) ^ 2 +*
3 ∗ max (nlength N) (nlength (previous hp1 t ∗ Z)) +
Suc Z ∗ (43 + 9 ∗ nlength (N + previous hp1 t ∗ Z + Z)) +
4 ∗ nllength [N + previous hp1 t ∗ Z..<N + previous hp1 t ∗ Z + Z] +
2 ∗ nlength (N + previous hp1 t ∗ Z)
shows *transforms tmT5 tps8 ttt tpsT5*
unfolding *tmT5-def* **by** (*tform tps*: *tps0 tpsT4-def tpsT5-def jk time*: *assms*)

definition *tps9 ≡ tps0*
[*j3 + 11* := (⌊*previous hp1 t*⌋$_N$, *1*),
j3 + 13 := (⌊*previous hp1 t = t*⌋$_B$, *1*),
j3 + 6 := (⌊*t − 1*⌋$_N$, *1*),
j3 + 7 := (⌊*0*⌋$_N$, *1*),
j3 + 12 := *nltape*
([*N + (t − 1) ∗ Z..<N + (t − 1) ∗ Z + Z*] @
(*if previous hp1 t ≠ t then* [*N + previous hp1 t ∗ Z..<N + previous hp1 t ∗ Z + Z*] *else* [])),
j3 + 8 := (⌊[]⌋, *1*)]

lemma *tm89* [*transforms-intros*]:
assumes *ttt = 37 + 26 ∗ (nlength (previous hp1 t) + nlength Z) ^ 2 +*
3 ∗ max (nlength N) (nlength (previous hp1 t ∗ Z)) +
Suc Z ∗ (43 + 9 ∗ nlength (N + previous hp1 t ∗ Z + Z)) +
4 ∗ nllength [N + previous hp1 t ∗ Z..<N + previous hp1 t ∗ Z + Z] +
2 ∗ nlength (N + previous hp1 t ∗ Z)
shows *transforms tm89 tps8 ttt tps9*
unfolding *tm89-def*
proof (*tform time*: *assms*)
have *tps8* ! (*j3 + 13*) = (⌊*previous hp1 t = t*⌋$_B$, *1*)
using *tps8-def jk* **by** *simp*
then have ∗: (*previous hp1 t ≠ t*) = (*read tps8* ! (*j3 + 13*) = □)
using *jk tps8-def read-ncontents-eq-0* **by** *force*
show *read tps8* ! (*j3 + 13*) = □ ⟹ *tps9 = tpsT5*
using ∗ *tps9-def tpsT5-def* **by** *simp*
have (⌊[]⌋, *1*) = *tps0* ! (*j3 + 8*)
using *jk tps0* **by** *simp*
then have *tps0*[*j3 + 8* := (⌊[]⌋, *1*)] = *tps0*
using *jk tps0* **by** (*metis list-update-id*)
then have *tps8 = tps0*
[*j3 + 11* := (⌊*previous hp1 t*⌋$_N$, *1*),
j3 + 13 := (⌊*previous hp1 t = t*⌋$_B$, *1*),
j3 + 6 := (⌊*t − 1*⌋$_N$, *1*),
j3 + 7 := (⌊*0*⌋$_N$, *1*),
j3 + 12 := *nltape* [*N + (t − 1) ∗ Z..<N + (t − 1) ∗ Z + Z*],
j3 + 8 := (⌊[]⌋, *1*)]
using *tps8-def jk tps0* **by** (*simp add*: *list-update-swap*[*of - j3 + 8*])
then show *read tps8* ! (*j3 + 13*) ≠ □ ⟹ *tps9 = tps8*

using ∗ *tps9-def* **by** *simp*
qed

definition *ttt10* ≡ *ttt8* + *37* + *26* ∗ (*nlength* (*previous hp1 t*) + *nlength Z*) ^ *2* +
3 ∗ *max* (*nlength N*) (*nlength* (*previous hp1 t* ∗ *Z*)) +
Suc Z ∗ (*43* + *9* ∗ *nlength* (*N* + *previous hp1 t* ∗ *Z* + *Z*)) +
4 ∗ *nllength* [*N* + *previous hp1 t* ∗ *Z*..<*N* + *previous hp1 t* ∗ *Z* + *Z*] +
2 ∗ *nlength* (*N* + *previous hp1 t* ∗ *Z*)

lemma *tm10* [*transforms-intros*]: *transforms tm10 tps0 ttt10 tps9*
unfolding *tm10-def* **by** (*tform tps*: *ttt10-def*)

definition *tps11* ≡ *tps0*
[*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*),
j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
j3 + *6* := (⌊*t*⌋$_N$, *1*),
j3 + *7* := (⌊*0*⌋$_N$, *1*),
j3 + *12* := *nltape*
([*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*] @
(*if previous hp1 t* ≠ *t then* [*N* + *previous hp1 t* ∗ *Z*..<*N* + *previous hp1 t* ∗ *Z* + *Z*] *else* [])),
j3 + *8* := (⌊[]⌋, *1*)]

lemma *tm11* [*transforms-intros*]:
assumes *ttt* = *ttt10* + *5* + *2* ∗ *nlength* (*t* − *1*)
shows *transforms tm11 tps0 ttt tps11*
unfolding *tm11-def* **by** (*tform tps*: *t*(*1*) *tps0 tps9-def tps11-def jk time*: *assms*)

definition *tps13* ≡ *tps0*
[*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*),
j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
j3 + *6* := (⌊*t*⌋$_N$, *1*),
j3 + *7* := (⌊*t* ∗ *Z*⌋$_N$, *1*),
j3 + *12* := *nltape*
([*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*] @
(*if previous hp1 t* ≠ *t then* [*N* + *previous hp1 t* ∗ *Z*..<*N* + *previous hp1 t* ∗ *Z* + *Z*] *else* [])),
j3 + *8* := (⌊[]⌋, *1*)]

lemma *tm13* [*transforms-intros*]:
assumes *ttt* = *ttt10* + *9* + *2* ∗ *nlength* (*t* − *1*) + *26* ∗ (*nlength t* + *nlength Z*) ^ *2*
shows *transforms tm13 tps0 ttt tps13*
unfolding *tm13-def*
proof (*tform tps*: *tps0 tps11-def tps13-def jk*)
show *ttt* = *ttt10* + *5* + *2* ∗ *nlength* (*t* − *1*) + (*4* + *26* ∗ (*nlength t* + *nlength Z*) ∗ (*nlength t* + *nlength Z*))
using *assms* **by** *simp* (*metis Groups.mult-ac*(*1*) *distrib-left power2-eq-square*)
show *tps13* = *tps11*[*j3* + *7* := (⌊*t* ∗ *Z*⌋$_N$, *1*)]
unfolding *tps13-def tps11-def* **by** (*simp add*: *list-update-swap*[*of j3*+*7*])
qed

definition *tps14* ≡ *tps0*
[*j3* + *11* := (⌊*previous hp1 t*⌋$_N$, *1*),
j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
j3 + *6* := (⌊*t*⌋$_N$, *1*),
j3 + *7* := (⌊*N* + *t* ∗ *Z*⌋$_N$, *1*),
j3 + *12* := *nltape*
([*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*] @
(*if previous hp1 t* ≠ *t then* [*N* + *previous hp1 t* ∗ *Z*..<*N* + *previous hp1 t* ∗ *Z* + *Z*] *else* [])),
j3 + *8* := (⌊[]⌋, *1*)]

lemma *tm14* [*transforms-intros*]:
assumes *ttt* = *ttt10* + *19* + *2* ∗ *nlength* (*t* − *1*) + *26* ∗ (*nlength t* + *nlength Z*) ^ *2* +
3 ∗ *max* (*nlength N*) (*nlength* (*t* ∗ *Z*))
shows *transforms tm14 tps0 ttt tps14*
unfolding *tm14-def*

proof (*tform tps*: *tps0 tps13-def tps14-def jk time*: *assms*)
show $tps14 = tps13[j3 + 7 := (\lfloor N + t * Z \rfloor_N, 1)]$
unfolding *tps14-def tps13-def* **by** (*simp add*: *list-update-swap*[*of j3+7*])
qed

definition $tps15 \equiv tps0$
$[j3 + 11 := (\lfloor previous\ hp1\ t \rfloor_N, 1),$
$j3 + 13 := (\lfloor previous\ hp1\ t = t \rfloor_B, 1),$
$j3 + 6 := (\lfloor t \rfloor_N, 1),$
$j3 + 7 := (\lfloor N + t * Z \rfloor_N, 1),$
$j3 + 12 := nltape$
$([N + (t - 1) * Z..<N + (t - 1) * Z + Z]$ @
$(if\ previous\ hp1\ t \neq t\ then\ [N + previous\ hp1\ t * Z..<N + previous\ hp1\ t * Z + Z]\ else\ [])),$
$j3 + 8 := (\lfloor [N + t * Z..<N + t * Z + Z] \rfloor_{NL}, 1)]$

lemma *tm15* [*transforms-intros*]:
assumes $ttt = ttt10 + 19 + 2 * nlength\ (t - 1) + 26 * (nlength\ t + nlength\ Z)\ \hat{}\ 2 +$
$3 * max\ (nlength\ N)\ (nlength\ (t * Z)) + Suc\ Z * (43 + 9 * nlength\ (N + t * Z + Z))$
shows *transforms tm15 tps0 ttt tps15*
unfolding *tm15-def*
proof (*tform tps*: *tps0 tps14-def tps15-def jk time*: *assms*)
show $tps14\ !\ (j3 + 8) = (\lfloor [] \rfloor_{NL}, 1)$
using *tps14-def jk nlcontents-Nil tps0* **by** *simp*
show $tps14\ !\ (j3 + 8 + 1) = (\lfloor 0 \rfloor_N, 1)$
using *tps14-def jk canrepr-0 tps0-sym* **by** *simp*
show $tps14\ !\ (j3 + 8 + 2) = (\lfloor 0 \rfloor_N, 1)$
using *tps14-def jk canrepr-0 tps0-sym* **by** *simp*
qed

definition $tps16 \equiv tps0$
$[j3 + 11 := (\lfloor previous\ hp1\ t \rfloor_N, 1),$
$j3 + 13 := (\lfloor previous\ hp1\ t = t \rfloor_B, 1),$
$j3 + 6 := (\lfloor t \rfloor_N, 1),$
$j3 + 7 := (\lfloor N + t * Z \rfloor_N, 1),$
$j3 + 12 := nltape$
$([N + (t - Suc\ 0) * Z..<N + (t - Suc\ 0) * Z + Z]$ @
$(if\ previous\ hp1\ t \neq t\ then\ [N + previous\ hp1\ t * Z..<N + previous\ hp1\ t * Z + Z]\ else\ [])$ @
$[N + t * Z..<N + t * Z + Z]),$
$j3 + 8 := (\lfloor [] \rfloor, 1)]$

lemma *tm16* [*transforms-intros*]:
assumes $ttt = ttt10 + 30 + 2 * nlength\ (t - 1) + 26 * (nlength\ t + nlength\ Z)\ \hat{}\ 2 +$
$3 * max\ (nlength\ N)\ (nlength\ (t * Z)) + Suc\ Z * (43 + 9 * nlength\ (N + t * Z + Z)) +$
$4 * nllength\ [N + t * Z..<N + t * Z + Z]$
shows *transforms tm16 tps0 ttt tps16*
unfolding *tm16-def* **by** (*tform tps*: *tps0 tps15-def tps16-def jk time*: *assms*)

definition $tps17 \equiv tps0$
$[j3 + 11 := (\lfloor 0 \rfloor_N, 1),$
$j3 + 13 := (\lfloor previous\ hp1\ t = t \rfloor_B, 1),$
$j3 + 6 := (\lfloor t \rfloor_N, 1),$
$j3 + 7 := (\lfloor N + t * Z \rfloor_N, 1),$
$j3 + 12 := nltape$
$([N + (t - Suc\ 0) * Z..<N + (t - Suc\ 0) * Z + Z]$ @
$(if\ previous\ hp1\ t \neq t\ then\ [N + previous\ hp1\ t * Z..<N + previous\ hp1\ t * Z + Z]\ else\ [])$ @
$[N + t * Z..<N + t * Z + Z]),$
$j3 + 8 := (\lfloor [] \rfloor, 1)]$

lemma *tm17* [*transforms-intros*]:
assumes $ttt = ttt10 + 40 + 2 * nlength\ (t - 1) + 26 * (nlength\ t + nlength\ Z)\ \hat{}\ 2 +$
$3 * max\ (nlength\ N)\ (nlength\ (t * Z)) + Suc\ Z * (43 + 9 * nlength\ (N + t * Z + Z)) +$
$4 * nllength\ [N + t * Z..<N + t * Z + Z] + 2 * nlength\ (previous\ hp1\ t)$
shows *transforms tm17 tps0 ttt tps17*

unfolding *tm17-def* **by** (*tform tps*: *tps0 tps16-def tps17-def jk time*: *jk assms*)

definition *tps18* ≡ *tps0*
[*j3* + *11* := ($\lfloor hp0\ !\ t \rfloor_N$, *1*),
j3 + *13* := ($\lfloor previous\ hp1\ t = t \rfloor_B$, *1*),
j3 + *6* := ($\lfloor t \rfloor_N$, *1*),
j3 + *7* := ($\lfloor N + t * Z \rfloor_N$, *1*),
j3 + *12* := *nltape*
([*N* + (*t* − *Suc 0*) ∗ *Z*..<*N* + (*t* − *Suc 0*) ∗ *Z* + *Z*] @
(*if previous hp1 t* ≠ *t then* [*N* + *previous hp1 t* ∗ *Z*..<*N* + *previous hp1 t* ∗ *Z* + *Z*] *else* []) @
[*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*]),
j3 + *8* := ($\lfloor [] \rfloor$, *1*)]

lemma *tm18* [*transforms-intros*]:
assumes *ttt* = *ttt10* + *66* + *2* ∗ *nlength* (*t* − *1*) + *26* ∗ (*nlength t* + *nlength Z*) ^ *2* +
3 ∗ *max* (*nlength N*) (*nlength* (*t* ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ *nlength* (*N* + *t* ∗ *Z* + *Z*)) +
4 ∗ *nllength* [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] + *2* ∗ *nlength* (*previous hp1 t*) +
21 ∗ $(nllength\ hp0)^2$
shows *transforms tm18 tps0 ttt tps18*
unfolding *tm18-def*
proof (*tform tps*: *tps0 tps18-def tps17-def jk time*: *assms*)
show *t* < *length hp0*
using *len-hp0 t* **by** *simp*
qed

definition *tps19* ≡ *tps0*
[*j3* + *11* := ($\lfloor hp0\ !\ t \rfloor_N$, *1*),
j3 + *13* := ($\lfloor previous\ hp1\ t = t \rfloor_B$, *1*),
j3 + *6* := ($\lfloor t \rfloor_N$, *1*),
j3 + *7* := ($\lfloor 0 \rfloor_N$, *1*),
j3 + *12* := *nltape*
([*N* + (*t* − *Suc 0*) ∗ *Z*..<*N* + (*t* − *Suc 0*) ∗ *Z* + *Z*] @
(*if previous hp1 t* ≠ *t then* [*N* + *previous hp1 t* ∗ *Z*..<*N* + *previous hp1 t* ∗ *Z* + *Z*] *else* []) @
[*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*]),
j3 + *8* := ($\lfloor [] \rfloor$, *1*)]

lemma *tm19* [*transforms-intros*]:
assumes *ttt* = *ttt10* + *76* + *2* ∗ *nlength* (*t* − *1*) + *26* ∗ (*nlength t* + *nlength Z*) ^ *2* +
3 ∗ *max* (*nlength N*) (*nlength* (*t* ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ *nlength* (*N* + *t* ∗ *Z* + *Z*)) +
4 ∗ *nllength* [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] + *2* ∗ *nlength* (*previous hp1 t*) +
21 ∗ $(nllength\ hp0)^2$ + *2* ∗ *nlength* (*N* + *t* ∗ *Z*)
shows *transforms tm19 tps0 ttt tps19*
unfolding *tm19-def*
by (*tform tps*: *tps0 tps18-def tps19-def jk time*: *assms*)

definition *tps20* ≡ *tps0*
[*j3* + *11* := ($\lfloor hp0\ !\ t \rfloor_N$, *1*),
j3 + *13* := ($\lfloor previous\ hp1\ t = t \rfloor_B$, *1*),
j3 + *6* := ($\lfloor t \rfloor_N$, *1*),
j3 + *7* := ($\lfloor hp0\ !\ t * G \rfloor_N$, *1*),
j3 + *12* := *nltape*
([*N* + (*t* − *Suc 0*) ∗ *Z*..<*N* + (*t* − *Suc 0*) ∗ *Z* + *Z*] @
(*if previous hp1 t* ≠ *t then* [*N* + *previous hp1 t* ∗ *Z*..<*N* + *previous hp1 t* ∗ *Z* + *Z*] *else* []) @
[*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*]),
j3 + *8* := ($\lfloor [] \rfloor$, *1*)]

definition *ttt20* ≡ *ttt10* + *80* + *2* ∗ *nlength* (*t* − *1*) + *26* ∗ (*nlength t* + *nlength Z*) ^ *2* +
3 ∗ *max* (*nlength N*) (*nlength* (*t* ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ *nlength* (*N* + *t* ∗ *Z* + *Z*)) +
4 ∗ *nllength* [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] + *2* ∗ *nlength* (*previous hp1 t*) +
21 ∗ $(nllength\ hp0)^2$ + *2* ∗ *nlength* (*N* + *t* ∗ *Z*) + *26* ∗ (*nlength* (*hp0* ! *t*) + *nlength G*) ^ *2*

lemma *tm20* [*transforms-intros*]: *transforms tm20 tps0 ttt20 tps20*
unfolding *tm20-def*

proof (*tform tps*: *tps0 tps19-def tps20-def jk*)
  show *tps20* = *tps19*[*j3* + *7* := (⌊*hp0* ! *t* ∗ *G*⌋$_N$, *1*)]
    unfolding *tps20-def tps19-def* **by** (*simp add*: *list-update-swap*[*of j3* + *7*])
  show *ttt20* = *ttt10* + *76* + *2* ∗ *nlength* (*t* − *1*) + *26* ∗ (*nlength t* + *nlength Z*)^2 +
      3 ∗ *max* (*nlength N*) (*nlength* (*t* ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ *nlength* (*N* + *t* ∗ *Z* + *Z*)) +
      4 ∗ *nllength* [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] + *2* ∗ *nlength* (*previous hp1 t*) +
      21 ∗ (*nllength hp0*)^2 + *2* ∗ *nlength* (*N* + *t* ∗ *Z*) +
      (*4* + *26* ∗ (*nlength* (*hp0* ! *t*) + *nlength G*) ∗ (*nlength* (*hp0* ! *t*) + *nlength G*))
    using *ttt20-def* **by** *simp* (*metis Groups.mult-ac*(*1*) *distrib-left power2-eq-square*)
qed

definition *tps21* ≡ *tps0*
  [*j3* + *11* := (⌊*hp0* ! *t*⌋$_N$, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
   j3 + *6* := (⌊*t*⌋$_N$, *1*),
   j3 + *7* := (⌊*hp0* ! *t* ∗ *G*⌋$_N$, *1*),
   j3 + *12* := *nltape*
      ([*N* + (*t* − *Suc 0*) ∗ *Z*..<*N* + (*t* − *Suc 0*) ∗ *Z* + *Z*] @
       (*if previous hp1 t* ≠ *t then* [*N* + *previous hp1 t* ∗ *Z*..<*N* + *previous hp1 t* ∗ *Z* + *Z*] *else* []) @
       [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*]),
   j3 + *8* := (⌊[*hp0* ! *t* ∗ *G*..<*hp0* ! *t* ∗ *G* + *G*]⌋$_{NL}$, *1*)]

lemma *tm21* [*transforms-intros*]:
  assumes *ttt* = *ttt20* + *Suc G* ∗ (*43* + *9* ∗ *nlength* (*hp0* ! *t* ∗ *G* + *G*))
  shows *transforms tm21 tps0 ttt tps21*
  unfolding *tm21-def*
proof (*tform tps*: *tps0 tps20-def tps21-def jk time*: *assms*)
  show *tps20* ! (*j3* + *8*) = (⌊[]⌋$_{NL}$, *1*)
    using *tps20-def jk nlcontents-Nil tps0* **by** *simp*
  show *tps20* ! (*j3* + *8* + *1*) = (⌊*0*⌋$_N$, *1*)
    using *tps20-def jk canrepr-0 tps0-sym* **by** *simp*
  show *tps20* ! (*j3* + *8* + *2*) = (⌊*0*⌋$_N$, *1*)
    using *tps20-def jk canrepr-0 tps0-sym* **by** *simp*
qed

abbreviation σ ≡
  [*N* + (*t* − *1*) ∗ *Z*..<*N* + (*t* − *1*) ∗ *Z* + *Z*] @
  (*if previous hp1 t* ≠ *t then* [*N* + *previous hp1 t* ∗ *Z*..<*N* + *previous hp1 t* ∗ *Z* + *Z*] *else* []) @
  [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] @
  [*hp0* ! *t* ∗ *G*..<*hp0* ! *t* ∗ *G* + *G*]

definition *tps22* ≡ *tps0*
  [*j3* + *11* := (⌊*hp0* ! *t*⌋$_N$, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
   j3 + *6* := (⌊*t*⌋$_N$, *1*),
   j3 + *7* := (⌊*hp0* ! *t* ∗ *G*⌋$_N$, *1*),
   j3 + *12* := *nltape* σ,
   j3 + *8* := (⌊[]⌋, *1*)]

lemma *tm22* [*transforms-intros*]:
  assumes *ttt* = *ttt20* + *11* + *Suc G* ∗ (*43* + *9* ∗ *nlength* (*hp0* ! *t* ∗ *G* + *G*)) +
    4 ∗ *nllength* [*hp0* ! *t* ∗ *G*..<*hp0* ! *t* ∗ *G* + *G*]
  shows *transforms tm22 tps0 ttt tps22*
  unfolding *tm22-def* **by** (*tform tps*: *tps0 tps21-def tps22-def jk time*: *assms*)

definition *tps23* ≡ *tps0*
  [*j3* + *11* := (⌊*hp0* ! *t*⌋$_N$, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
   j3 + *6* := (⌊*t*⌋$_N$, *1*),
   j3 + *7* := (⌊*0*⌋$_N$, *1*),
   j3 + *12* := *nltape* σ,
   j3 + *8* := (⌊[]⌋, *1*)]

lemma *tm23* [*transforms-intros*]:
  assumes *ttt* = *ttt20* + *21* + *Suc G* ∗ (*43* + *9* ∗ *nlength* (*hp0* ! *t* ∗ *G* + *G*)) +
    4 ∗ *nllength* [*hp0* ! *t* ∗ *G*..<*hp0* ! *t* ∗ *G* + *G*] + *2* ∗ *nlength* (*hp0* ! *t* ∗ *G*)
  shows *transforms tm23 tps0 ttt tps23*
  unfolding *tm23-def* **by** (*tform tps*: *tps0 tps22-def tps23-def jk time*: *assms*)

definition *tps24* ≡ *tps0*
  [*j3* + *11* := (⌊[]⌋, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
   j3 + *6* := (⌊*t*⌋$_N$, *1*),
   j3 + *7* := (⌊*0*⌋$_N$, *1*),
   j3 + *12* := *nltape* σ,
   j3 + *8* := (⌊[]⌋, *1*)]

lemma *tm24* [*transforms-intros*]:
  assumes *ttt* = *ttt20* + *28* + *Suc G* ∗ (*43* + *9* ∗ *nlength* (*hp0* ! *t* ∗ *G* + *G*)) +
    4 ∗ *nllength* [*hp0* ! *t* ∗ *G*..<*hp0* ! *t* ∗ *G* + *G*] + *2* ∗ *nlength* (*hp0* ! *t* ∗ *G*) +
    2 ∗ *nlength* (*hp0* ! *t*)
  shows *transforms tm24 tps0 ttt tps24*
  unfolding *tm24-def*
proof (*tform tps*: *tps0 tps23-def tps24-def jk assms*)
  show *proper-symbols* (*canrepr* (*hp0* ! *t*))
    using *proper-symbols-canrepr* **by** *simp*
qed

definition *tps25* ≡ *tps0*
  [*j3* + *11* := (⌊[]⌋, *1*),
   j3 + *13* := (⌊*previous hp1 t* = *t*⌋$_B$, *1*),
   j3 + *6* := (⌊*t*⌋$_N$, *1*),
   j3 + *7* := (⌊*0*⌋$_N$, *1*),
   j3 + *12* := (⌊σ⌋$_{NL}$, *1*),
   j3 + *8* := (⌊[]⌋, *1*)]

lemma *tm25* [*transforms-intros*]:
  assumes *ttt* = *ttt20* + *31* + *Suc G* ∗ (*43* + *9* ∗ *nlength* (*hp0* ! *t* ∗ *G* + *G*)) +
    4 ∗ *nllength* [*hp0* ! *t* ∗ *G*..<*hp0* ! *t* ∗ *G* + *G*] + *2* ∗ *nlength* (*hp0* ! *t* ∗ *G*) +
    2 ∗ *nlength* (*hp0* ! *t*) + *nllength* σ
  shows *transforms tm25 tps0 ttt tps25*
  unfolding *tm25-def*
proof (*tform tps*: *tps0 tps24-def tps25-def jk assms*)
  have ∗: *tps24* ! (*j3* + *12*) = *nltape* σ
    using *tps24-def jk* **by** *simp*
  then show *clean-tape* (*tps24* ! (*j3* + *12*))
    using *clean-tape-nlcontents* **by** *simp*
  have *tps25* = *tps24*[*j3* + *12* := *nltape* σ |#=| *1*]
    unfolding *tps25-def tps24-def* **by** (*simp add*: *list-update-swap*)
  then show *tps25* = *tps24*[*j3* + *12* := *tps24* ! (*j3* + *12*) |#=| *1*]
    using ∗ **by** *simp*
qed

definition *tpsE1* ≡ *tps0*
  [*j3* + *11* := (⌊[]⌋, *1*),
   j3 + *13* := (⌊[]⌋, *1*),
   j3 + *6* := (⌊*t*⌋$_N$, *1*),
   j3 + *7* := (⌊*0*⌋$_N$, *1*),
   j3 + *12* := (⌊σ⌋$_{NL}$, *1*),
   j3 + *8* := (⌊[]⌋, *1*)]

lemma *tmE1*:
  assumes *ttt* = *7* + *2* ∗ *nlength* (*if previous hp1 t* = *t then 1 else 0*)
  shows *transforms tmE1 tps25 ttt tpsE1*
  unfolding *tmE1-def*
proof (*tform tps*: *tps0 tps25-def tpsE1-def jk assms*)

show *proper-symbols* (*canrepr* (*if previous hp1 t* = *t then 1 else 0*))
  using *proper-symbols-canrepr* **by** *simp*
qed

lemma *tmE1′* [*transforms-intros*]:
 assumes *ttt* = *9*
 shows *transforms tmE1 tps25 ttt tpsE1*
 using *tmE1 assms transforms-monotone* **by** (*simp add*: *nlength-le-n*)

definition *tpsE2* ≡ *tps0*
 [*j3* + *11* := *nlltape′* (*formula-n* (*relabel* σ ψ')) *0*,
  j3 + *13* := (⌊[]⌋, *1*),
  j3 + *6* := ($\lfloor t \rfloor_N$, *1*),
  j3 + *7* := ($\lfloor 0 \rfloor_N$, *1*),
  j3 + *12* := ($\lfloor \sigma \rfloor_{NL}$, *1*),
  j3 + *8* := (⌊[]⌋, *1*)]

lemma *tmE2* [*transforms-intros*]:
 assumes *ttt* = *16* + *108* ∗ (*nlllength* (*formula-n* ψ'))^2 ∗ (*3* + (*nllength* σ)^2)
  and *previous hp1 t* = *t*
 shows *transforms tmE2 tps25 ttt tpsE2*
 unfolding *tmE2-def*
proof (*tform tps*: *tps0 tpsE1-def tpsE2-def jk assms time*: *assms*(*1*))
 let *?sigma* = [*N* + (*t* − *Suc 0*) ∗ *Z*..<*N* + (*t* − *Suc 0*) ∗ *Z* + *Z*] @
     [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] @ [*hp0* ! *t* ∗ *G*..<*hp0* ! *t* ∗ *G* + *G*]
 show *variables* ψ' ⊆ {..<*length ?sigma*}
  using *assms*(*2*) *psi′*(*1*) **by** *auto*
 show *tpsE1* ! (*j3* + *11*) = ($\lfloor [] \rfloor_{NLL}$, *1*)
  using *tpsE1-def jk nllcontents-Nil* **by** *simp*
 show *tpsE1* ! (*j3* + *11* + *1*) = ($\lfloor ?sigma \rfloor_{NL}$, *1*)
  using *assms*(*2*) *tpsE1-def jk* **by** (*simp add*: *add.commute*[*of j3 12*])
 show *tpsE1* ! (*j3* + *11* + *2*) = ($\lfloor [] \rfloor_{NL}$, *1*)
  using *tpsE1-def jk nlcontents-Nil* **by** (*simp add*: *add.commute*[*of j3 13*])
 show *tpsE1* ! (*j3* + *11* + *3*) = ($\lfloor [] \rfloor_{NL}$, *1*)
  using *tpsE1-def jk tps0-sym nlcontents-Nil* **by** *simp*
 show *tpsE1* ! (*j3* + *11* + *4*) = ($\lfloor 0 \rfloor_N$, *1*)
  using *tpsE1-def jk tps0-sym canrepr-0* **by** *simp*
 show *tpsE1* ! (*j3* + *11* + *5*) = ($\lfloor 0 \rfloor_N$, *1*)
  using *tpsE1-def jk tps0-sym canrepr-0* **by** *simp*
 show *tpsE2* = *tpsE1*[*j3* + *11* := *nlltape′* (*formula-n* (*relabel ?sigma* ψ')) *0*]
  unfolding *tpsE2-def tpsE1-def* **using** *assms*(*2*) **by** (*simp add*: *list-update-swap*[*of j3*+*11*])
qed

definition *tpsTT1* ≡ *tps0*
 [*j3* + *11* := *nlltape′* (*formula-n* (*relabel* σ ψ)) *0*,
  j3 + *13* := (⌊[]⌋, *1*),
  j3 + *6* := ($\lfloor t \rfloor_N$, *1*),
  j3 + *7* := ($\lfloor 0 \rfloor_N$, *1*),
  j3 + *12* := ($\lfloor \sigma \rfloor_{NL}$, *1*),
  j3 + *8* := (⌊[]⌋, *1*)]

lemma *tmTT1* [*transforms-intros*]:
 assumes *ttt* = *7* + *108* ∗ (*nlllength* (*formula-n* ψ))^2 ∗ (*3* + (*nllength* σ)^2)
  and *previous hp1 t* ≠ *t*
 shows *transforms tmTT1 tps25 ttt tpsTT1*
 unfolding *tmTT1-def*
proof (*tform tps*: *tps0 tps25-def tpsTT1-def jk time*: *assms*(*1*))
 let *?sigma* = [*N* + (*t* − *Suc 0*) ∗ *Z*..<*N* + (*t* − *Suc 0*) ∗ *Z* + *Z*] @
     (*if previous hp1 t* ≠ *t*
      then [*N* + *previous hp1 t* ∗ *Z*..<*N* + *previous hp1 t* ∗ *Z* + *Z*]
      else []) @
     [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] @ [*hp0* ! *t* ∗ *G*..<*hp0* ! *t* ∗ *G* + *G*]
 show *variables* ψ ⊆ {..<*length ?sigma*}

using *assms*(*2*) *psi*(*1*) **by** *auto*
show *tps25* ! (*j3* + *11*) = ($\lfloor[]\rfloor_{NLL}$, *1*)
using *tps25-def jk nllcontents-Nil* **by** *simp*
show *tps25* ! (*j3* + *11* + *1*) = ($\lfloor ?sigma\rfloor_{NL}$, *1*)
using *tps25-def jk* **by** (*simp add*: *add.commute*[*of j3 12*])
show *tps25* ! (*j3* + *11* + *2*) = ($\lfloor[]\rfloor_{NL}$, *1*)
using *tps25-def jk canrepr-0 nlcontents-Nil assms*(*2*) **by** (*simp add*: *add.commute*[*of j3 13*])
show *tps25* ! (*j3* + *11* + *3*) = ($\lfloor[]\rfloor_{NL}$, *1*)
using *tps25-def jk tps0-sym nlcontents-Nil* **by** *simp*
show *tps25* ! (*j3* + *11* + *4*) = ($\lfloor 0\rfloor_{N}$, *1*)
using *tps25-def jk tps0-sym canrepr-0* **by** *simp*
show *tps25* ! (*j3* + *11* + *5*) = ($\lfloor 0\rfloor_{N}$, *1*)
using *tps25-def jk tps0-sym canrepr-0* **by** *simp*
show *tpsTT1* = *tps25*[*j3* + *11* := *nlltape'* (*formula-n* (*relabel ?sigma* ψ)) *0*]
using *tpsTT1-def tps25-def assms*(*2*) *canrepr-0* **by** (*simp add*: *list-update-swap*[*of j3*+*11*])
qed

definition *tps26* ≡ *tps0*
[*j3* + *11* := *nlltape'* (*formula-n* (*relabel* σ (*if previous hp1 t* = *t then* ψ' *else* ψ))) *0*,
j3 + *13* := ($\lfloor[]\rfloor$, *1*),
j3 + *6* := ($\lfloor t\rfloor_{N}$, *1*),
j3 + *7* := ($\lfloor 0\rfloor_{N}$, *1*),
j3 + *12* := ($\lfloor\sigma\rfloor_{NL}$, *1*),
j3 + *8* := ($\lfloor[]\rfloor$, *1*)]

lemma *nlllength-psi*: *nlllength* (*formula-n* ψ) ≤ *24* ∗ *Z* ^ *2* ∗ *2* ^ (*4*∗*Z*)
proof −
let *?V* = *3* ∗ *Z* + *G*
have ⋀*v*. *v* ∈ *variables* ψ ⟹ *v* ≤ *?V*
using *psi*(*1*) **by** *auto*
then have *nlllength* (*formula-n* ψ) ≤ *fsize* ψ ∗ *Suc* (*Suc* (*nlength ?V*)) + *length* ψ
using *nlllength-formula-n* **by** *simp*
also have ... ≤ *fsize* ψ ∗ *Suc* (*Suc* (*nlength ?V*)) + *2* ^ *?V*
using *psi* **by** *simp*
also have ... ≤ *?V* ∗ *2* ^ *?V* ∗ *Suc* (*Suc* (*nlength ?V*)) + *2* ^ *?V*
using *psi*(*2*) **by** (*metis add-le-mono1 mult.commute mult-le-mono2*)
also have ... ≤ *4*∗*Z* ∗ *2* ^ *?V* ∗ *Suc* (*Suc* (*nlength ?V*)) + *2* ^ *?V*
using *Z* **by** *simp*
also have ... ≤ *4*∗*Z* ∗ *2* ^ (*4*∗*Z*) ∗ *Suc* (*Suc* (*nlength ?V*)) + *2* ^ *?V*
proof −
have *?V* ≤ *4* ∗ *Z*
using *Z* **by** *simp*
then have (*2*::*nat*) ^ *?V* ≤ *2* ^ (*4*∗*Z*)
by *simp*
then show *?thesis*
using *add-le-mono le-refl mult-le-mono* **by** *presburger*
qed
also have ... ≤ *4*∗*Z* ∗ (*2*::*nat*) ^ (*4*∗*Z*) ∗ *Suc* (*Suc* (*nlength* (*4*∗*Z*))) + *2* ^ *?V*
using *Z nlength-mono* **by** *simp*
also have ... ≤ *4*∗*Z* ∗ (*2*::*nat*) ^ (*4*∗*Z*) ∗ *Suc* (*Suc* (*4*∗*Z*)) + *2* ^ *?V*
using *nlength-le-n* **by** *simp*
also have ... ≤ *4*∗*Z* ∗ *2* ^ (*4*∗*Z*) ∗ *Suc* (*Suc* (*4*∗*Z*)) + *2* ^ (*4*∗*Z*)
using *Z* **by** *simp*
also have ... ≤ *4*∗*Z* ∗ *2* ^ (*4*∗*Z*) ∗ (*5*∗*Z*) + *2* ^ (*4*∗*Z*)
using *Z G* **by** *simp*
also have ... ≤ *4*∗*Z* ∗ *2* ^ (*4*∗*Z*) ∗ (*6*∗*Z*)
using *Z G* **by** *simp*
also have ... = *24* ∗ *Z* ^ *2* ∗ *2* ^ (*4*∗*Z*)
by *algebra*
finally show *?thesis* **.**
qed

lemma *nlllength-psi'*: *nlllength* (*formula-n* ψ') ≤ *24* ∗ *Z* ^ *2* ∗ *2* ^ (*4*∗*Z*)

```
proof -
  let ?V = 2 * Z + G
  have ⋀v. v ∈ variables ψ' ⟹ v ≤ ?V
    using psi'(1) by auto
  then have nlllength (formula-n ψ') ≤ fsize ψ' * Suc (Suc (nlength ?V)) + length ψ'
    using nlllength-formula-n by simp
  also have ... ≤ fsize ψ' * Suc (Suc (nlength ?V)) + 2 ^ ?V
    using psi' by simp
  also have ... ≤ ?V * 2 ^ ?V * Suc (Suc (nlength ?V)) + 2 ^ ?V
    using psi'(2) by (metis add-le-mono1 mult.commute mult-le-mono2)
  also have ... ≤ 4*Z * 2 ^ ?V * Suc (Suc (nlength ?V)) + 2 ^ ?V
    using Z by simp
  also have ... ≤ 4*Z * 2 ^ (4*Z) * Suc (Suc (nlength ?V)) + 2 ^ ?V
  proof -
    have ?V ≤ 4 * Z
      using Z by simp
    then have (2::nat) ^ ?V ≤ 2 ^ (4*Z)
      by simp
    then show ?thesis
      using add-le-mono le-refl mult-le-mono by presburger
  qed
  also have ... ≤ 4*Z * (2::nat) ^ (4*Z) * Suc (Suc (nlength (4*Z))) + 2 ^ ?V
    using Z nlength-mono by simp
  also have ... ≤ 4*Z * (2::nat) ^ (4*Z) * Suc (Suc (4*Z)) + 2 ^ ?V
    using nlength-le-n by simp
  also have ... ≤ 4*Z * 2 ^ (4*Z) * Suc (Suc (4*Z)) + 2 ^ (4*Z)
    using Z by simp
  also have ... ≤ 4*Z * 2 ^ (4*Z) * (5*Z) + 2 ^ (4*Z)
    using Z G by simp
  also have ... ≤ 4*Z * 2 ^ (4*Z) * (6*Z)
    using Z G by simp
  also have ... = 24 * Z ^ 2 * 2 ^ (4*Z)
    by algebra
  finally show ?thesis .
qed

lemma tm2526:
  assumes ttt = 17 + 108 * (24 * Z ^ 2 * 2 ^ (4*Z))² * (3 + (nllength σ)²)
  shows transforms tm2526 tps25 ttt tps26
  unfolding tm2526-def
proof (tform)
  have *: tps25 ! (j3 + 13) = (⌊previous hp1 t = t⌋_B, 1)
    using tps25-def jk by simp
  then have **: (previous hp1 t ≠ t) = (read tps25 ! (j3 + 13) = □)
    using jk tps25-def read-ncontents-eq-0 by force
  show read tps25 ! (j3 + 13) = □ ⟹ previous hp1 t ≠ t
    using ** by simp
  show read tps25 ! (j3 + 13) ≠ □ ⟹ previous hp1 t = t
    using ** by simp
  show read tps25 ! (j3 + 13) = □ ⟹ tps26 = tpsTT1
    using tps26-def tpsTT1-def ** by presburger
  show read tps25 ! (j3 + 13) ≠ □ ⟹ tps26 = tpsE2
    using tps26-def tpsE2-def ** by presburger

  let ?tT = 7 + 108 * (nlllength (formula-n ψ))² * (3 + (nllength σ)²)
  let ?tF = 16 + 108 * (nlllength (formula-n ψ'))² * (3 + (nllength σ)²)
  have ?tT + 2 ≤ 9 + 108 * (24 * Z ^ 2 * 2 ^ (4*Z))² * (3 + (nllength σ)²)
    using nlllength-psi linear-le-pow by simp
  also have ... ≤ ttt
    using assms by simp
  finally show ?tT + 2 ≤ ttt .
  show ?tF + 1 ≤ ttt
    using nlllength-psi' assms linear-le-pow by simp
```

qed

lemma *nllength-σ*: *nllength* $\sigma \leq 12 * T' * Z\hat{~}2 + 4 * Z * N$
proof −
  have $n \leq N + T' * Z + Z$ **if** $n \in set\ \sigma$ **for** n
  proof −
    consider
        $n \in set\ [N + (t - 1) * Z..<N + (t - 1) * Z + Z]$
      | $n \in set\ [N + previous\ hp1\ t * Z..<N + previous\ hp1\ t * Z + Z]$
      | $n \in set\ [N + t * Z..<N + t * Z + Z]$
      | $n \in set\ [hp0\ !\ t * G..<hp0\ !\ t * G + G]$
      using ‹$n \in set\ \sigma$› **by** *auto*
    then show *?thesis*
    proof (*cases*)
      case *1*
      then have $n \leq N + (t - 1) * Z + Z$
        by *simp*
      also have ... $\leq N + T' * Z + Z$
        using *t*(*2*) **by** *auto*
      finally show *?thesis*
        by *simp*
    next
      case *2*
      then have $n \leq N + (previous\ hp1\ t) * Z + Z$
        by *simp*
      also have ... $\leq N + t * Z + Z$
        by (*simp add*: *previous-hp1-le*)
      also have ... $\leq N + T' * Z + Z$
        using *t*(*2*) **by** *simp*
      finally show *?thesis*
        by *simp*
    next
      case *3*
      then have $n \leq N + t * Z + Z$
        by *simp*
      also have ... $\leq N + T' * Z + Z$
        using *t*(*2*) **by** *auto*
      finally show *?thesis*
        by *simp*
    next
      case *4*
      then have $n \leq N + (hp0\ !\ t) * G + G$
        by *simp*
      also have ... $\leq N + T' * G + G$
        using *t len-hp0 hp0* **by** *simp*
      also have ... $\leq N + T' * Z + Z$
        using *Z* **by** *simp*
      finally show *?thesis*
        by *simp*
    qed
  qed
  then have *nllength* $\sigma \leq Suc\ (N + T' * Z + Z) * (length\ \sigma)$
    using *nllength-le*[*of* σ $N + T' * Z + Z$] **by** *simp*
  also have ... $\leq Suc\ (N + T' * Z + Z) * (3 * Z + G)$
  proof −
    have *length* $\sigma \leq 3 * Z + G$
      by *simp*
    then show *?thesis*
      using *mult-le-mono2* **by** *presburger*
  qed
  also have ... $\leq Suc\ (N + T' * Z + Z) * (3 * Z + Z)$
    using *Z* **by** *simp*
  also have ... $= 4 * Z * Suc\ (N + T' * Z + Z)$

by *simp*
also have ... $= 4 * Z + 4 * Z * (N + T' * Z + Z)$
by *simp*
also have ... $= 4 * Z + 4 * Z * N + 4 * T' * Z\hat{}2 + 4 * Z\hat{}2$
by *algebra*
also have ... $\leq 4 * Z\hat{}2 + 4 * Z * N + 4 * T' * Z\hat{}2 + 4 * Z\hat{}2$
using *linear-le-pow* **by** *simp*
also have ... $= 8 * Z\hat{}2 + 4 * Z * N + 4 * T' * Z\hat{}2$
by *simp*
also have ... $\leq 8 * T' * Z\hat{}2 + 4 * Z * N + 4 * T' * Z\hat{}2$
using *t* **by** *simp*
also have ... $= 12 * T' * Z\hat{}2 + 4 * Z * N$
by *simp*
finally show *?thesis* **.**
qed

lemma *tm2526'* [*transforms-intros*]:
assumes $ttt = 17 + 108 * (24 * Z \hat{} 2 * 2 \hat{} (4{*}Z))^2 * (3 + (12 * T' * Z\hat{}2 + 4 * Z * N)^2)$
shows *transforms tm2526 tps25 ttt tps26*
proof $-$
have $17 + 108 * (24 * Z \hat{} 2 * 2 \hat{} (4{*}Z))^2 * (3 + (nllength\ \sigma)^2) \leq ttt$
using *assms nllength-σ* **by** *simp*
then show *?thesis*
using *tm2526 transforms-monotone* **by** *simp*
qed

lemma *tm26* [*transforms-intros*]:
assumes $ttt = ttt20 + 31 + Suc\ G * (43 + 9 * nlength\ (hp0\ !\ t * G + G)) +$
$4 * nllength\ [hp0\ !\ t * G..<hp0\ !\ t * G + G] + 2 * nlength\ (hp0\ !\ t * G) +$
$2 * nlength\ (hp0\ !\ t) + nllength\ \sigma +$
$17 + 108 * (24 * Z \hat{} 2 * 2 \hat{} (4{*}Z))^2 * (3 + (12 * T' * Z\hat{}2 + 4 * Z * N)^2)$
shows *transforms tm26 tps0 ttt tps26*
unfolding *tm26-def* **by** (*tform tps*: *assms*)

definition $tps27 \equiv tps0$
$[j3 + 11 := nlltape'\ (formula\text{-}n\ (relabel\ \sigma\ (if\ previous\ hp1\ t = t\ then\ \psi'\ else\ \psi)))\ 0,$
$j3 + 13 := (\lfloor[]\rfloor, 1),$
$j3 + 6 := (\lfloor t\rfloor_N, 1),$
$j3 + 7 := (\lfloor 0\rfloor_N, 1),$
$j3 + 12 := (\lfloor[]\rfloor, 1),$
$j3 + 8 := (\lfloor[]\rfloor, 1)]$

lemma *tm27*:
assumes $ttt = ttt20 + 38 + Suc\ G * (43 + 9 * nlength\ (hp0\ !\ t * G + G)) +$
$4 * nllength\ [hp0\ !\ t * G..<hp0\ !\ t * G + G] + 2 * nlength\ (hp0\ !\ t * G) +$
$2 * nlength\ (hp0\ !\ t) + 3 * nllength\ \sigma +$
$17 + 108 * (24 * Z \hat{} 2 * 2 \hat{} (4{*}Z))^2 * (3 + (12 * T' * Z\hat{}2 + 4 * Z * N)^2)$
shows *transforms tm27 tps0 ttt tps27*
unfolding *tm27-def*
proof (*tform tps*: *tps0 tps26-def tps27-def jk*)
let $?zs = numlist\ \sigma$
show $tps26 ::: (j3 + 12) = \lfloor ?zs\rfloor$
using *tps26-def jk nlcontents-def* **by** *simp*
show *proper-symbols ?zs*
using *proper-symbols-numlist* **by** *simp*
show $ttt = ttt20 + 31 + Suc\ G * (43 + 9 * nlength\ (hp0\ !\ t * G + G)) +$
$4 * nllength\ [hp0\ !\ t * G..<hp0\ !\ t * G + G] + 2 * nlength\ (hp0\ !\ t * G) +$
$2 * nlength\ (hp0\ !\ t) + nllength\ \sigma +$
$17 + 108 * (24 * Z^2 * 2 \hat{} (4 * Z))^2 * (3 + (12 * T' * Z^2 + 4 * Z * N)^2) +$
$(tps26 :\#: (j3 + 12) + 2 * length\ (numlist\ \sigma) + 6)$
using *tps26-def jk nllength-def assms* **by** *simp*
qed

```
definition tps27' ≡ tps0
  [j3 + 11 := nlltape' (formula-n (relabel σ (if previous hp1 t = t then ψ' else ψ))) 0]

lemma tps27': tps27 = tps27'
proof −
  have 1: tps0[j3 + 13 := (⌊[]⌋, Suc 0)] = tps0
    using list-update-id[of tps0 j3+13] jk tps0 by simp
  have 2: tps0[j3 + 12 := (⌊[]⌋, Suc 0)] = tps0
    using list-update-id[of tps0 j3+12] jk tps0 by simp
  have 3: tps0[j3 + 8 := (⌊[]⌋, Suc 0)] = tps0
    using list-update-id[of tps0 j3+8] jk tps0 by simp
  have 4: tps0[j3 + 7 := (⌊0⌋N, Suc 0)] = tps0
    using list-update-id[of tps0 j3+7] canrepr-0 jk tps0 by simp
  have 5: tps0[j3 + 6 := (⌊t⌋N, Suc 0)] = tps0
    using list-update-id[of tps0 j3+6] jk tps0 by simp
  show ?thesis
    unfolding tps27-def tps27'-def
    using tps0
    by (simp split del: if-split add:
      list-update-swap[of - j3 + 13] 1
      list-update-swap[of - j3 + 12] 2
      list-update-swap[of - j3 + 8] 3
      list-update-swap[of - j3 + 7] 4
      list-update-swap[of - j3 + 6] 5)
qed

definition ttt27 = ttt20 + 38 + Suc G ∗ (43 + 9 ∗ nlength (hp0 ! t ∗ G + G)) +
    4 ∗ nllength [hp0 ! t ∗ G..<hp0 ! t ∗ G + G] + 2 ∗ nlength (hp0 ! t ∗ G) +
    2 ∗ nlength (hp0 ! t) + 3 ∗ nllength σ +
    17 + 108 ∗ (24 ∗ Z ^ 2 ∗ 2 ^ (4∗Z))^2 ∗ (3 + (12 ∗ T' ∗ Z^2 + 4 ∗ Z ∗ N)^2)

lemma tm27' [transforms-intros]: transforms tm27 tps0 ttt27 tps27'
  using ttt27-def tm27 nllength-σ tps27' transforms-monotone by simp

definition tps28 ≡ tps0
  [1 := nlltape (nss @ formula-n (relabel σ (if previous hp1 t = t then ψ' else ψ))),
   j3 + 11 := (⌊[]⌋, 1)]

lemma tm28:
  assumes ttt = ttt27 + (11 + 4 ∗ nlllength (formula-n (relabel σ (if previous hp1 t = t then ψ' else ψ))))
  shows transforms tm28 tps0 ttt tps28
  unfolding tm28-def by (tform tps: tps0(1) tps0 tps27'-def tps28-def jk time: ttt27-def assms)

lemma nlllength-relabel-chi-t:
  nlllength (formula-n (relabel σ (if previous hp1 t = t then ψ' else ψ))) ≤
     Suc (nllength σ) ∗ 24 ∗ Z ^ 2 ∗ 2 ^ (4∗Z)
  (is nlllength (formula-n (relabel σ ?phi)) ≤ -)
proof −
  have variables ?phi ⊆ {..<length σ}
  proof (cases previous hp1 t = t)
    case True
    then show ?thesis
      using psi'(1) by auto
  next
    case False
    then show ?thesis
      using psi(1) by auto
  qed
  then have nlllength (formula-n (relabel σ ?phi)) ≤ Suc (nllength σ) ∗ nlllength (formula-n ?phi)
    using nlllength-relabel-variables by simp
  moreover have nlllength (formula-n ?phi) ≤ 24 ∗ Z ^ 2 ∗ 2 ^ (4∗Z)
    using nlllength-psi nlllength-psi' by (cases previous hp1 t = t) simp-all
  ultimately have nlllength (formula-n (relabel σ ?phi)) ≤ Suc (nllength σ) ∗ (24 ∗ Z ^ 2 ∗ 2 ^ (4∗Z))
```

by (*meson le-trans mult-le-mono2*)
then show *?thesis*
by *linarith*
qed

definition *tps28′* ≡ *tps0*
[*1* := *nlltape* (*nss* @ *formula-n* (*relabel* σ (*if previous hp1 t* = *t then* ψ′ *else* ψ)))]

lemma *tps28′*: *tps28′* = *tps28*
unfolding *tps28′-def tps28-def* **using** *tps0 list-update-id*[*of tps0 j3*+*11*]
by (*simp add*: *list-update-swap*[*of - j3* + *11*])

lemma *tm28′* [*transforms-intros*]:
assumes *ttt* = *ttt27* + (*11* + *4* ∗ (*Suc* (*nllength* σ) ∗ *24* ∗ *Z* ^ *2* ∗ *2* ^ (*4*∗*Z*)))
shows *transforms tm28 tps0 ttt tps28′*
using *assms tm28 tps28′ nlllength-relabel-chi-t transforms-monotone* **by** *simp*

definition *tps29* ≡ *tps0*
[*1* := *nlltape* (*nss* @ *formula-n* (*relabel* σ (*if previous hp1 t* = *t then* ψ′ *else* ψ))),
j3 + *6* := (⌊*Suc t*⌋$_N$, *1*)]

lemma *tm29* [*transforms-intros*]:
assumes *ttt* = *ttt27* + *16* + *4* ∗ (*Suc* (*nllength* σ) ∗ *24* ∗ *Z* ^ *2* ∗ *2* ^ (*4*∗*Z*)) + *2* ∗ *nlength t*
shows *transforms tm29 tps0 ttt tps29*
unfolding *tm29-def* **by** (*tform tps*: *assms tps0 tps28′-def tps29-def jk*)

definition *tps30* ≡ *tps0*
[*1* := *nlltape* (*nss* @ *formula-n* (*relabel* σ (*if previous hp1 t* = *t then* ψ′ *else* ψ))),
j3 + *6* := (⌊*Suc t*⌋$_N$, *1*),
j3 + *3* := (⌊*T* − *1*⌋$_N$, *1*)]

lemma *tm30*:
assumes *ttt* = *ttt27* + *24* + *4* ∗ (*Suc* (*nllength* σ) ∗ *24* ∗ *Z* ^ *2* ∗ *2* ^ (*4*∗*Z*)) + *2* ∗ *nlength t* + *2* ∗ *nlength T*
shows *transforms tm30 tps0 ttt tps30*
unfolding *tm30-def* **by** (*tform tps*: *assms tps0 tps29-def tps30-def jk*)

Some helpers for bounding the running time:

lemma *pow4-le-2pow4*: *z* ^ *4* ≤ *2* ^ (*4*∗*z*) **for** *z* :: *nat*
proof −
have *z* ^ *4* = (*z* ^ *2*) ^ *2*
by *simp*
also have ... ≤ (*2*^(*2*∗*z*)) ^ *2*
using *pow2-le-2pow2 power-mono* **by** *blast*
also have ... = *2*^(*4*∗*z*)
by (*metis mult.commute mult-2-right numeral-Bit0 power-mult*)
finally show *?thesis* **.**
qed

lemma *pow8-le-2pow8*: *z* ^ *8* ≤ *2* ^ (*8*∗*z*) **for** *z* :: *nat*
proof −
have *z* ^ *8* = (*z* ^ *2*) ^ *4*
by *simp*
also have ... ≤ (*2*^(*2*∗*z*)) ^ *4*
using *pow2-le-2pow2 power-mono* **by** *blast*
also have ... = *2*^(*8*∗*z*)
by (*metis mult.commute mult-2-right numeral-Bit0 power-mult*)
finally show *?thesis* **.**
qed

lemma *Z-sq-le*: *Z*^*2* ≤ *2*^(*16*∗*Z*)
proof −
have *Z*^*2* ≤ *2*^(*2*∗*Z*)
using *pow2-le-2pow2*[*of Z*] **by** *simp*

also have ... $\leq 2\hat{}(16*Z)$
by *simp*
finally show $Z\hat{}2 \leq 2\hat{}(16*Z)$ **.**
qed

lemma *time-bound*:
$ttt27 + 24 + 4 * (Suc\ (nllength\ \sigma) * 24 * Z \hat{} 2 * 2 \hat{} (4*Z)) + 2 * nlength\ t + 2 * nlength\ T \leq 16114765 * 2\hat{}(16*Z) * N\hat{}6$
proof $-$
have *sum-sq*: $(a + b) \hat{} 2 \leq Suc\ (2 * b) * a \hat{} 2 + b \hat{} 2$ **for** $a\ b :: nat$
proof $-$
have $(a + b) \hat{} 2 = a \hat{} 2 + 2 * a * b + b \hat{} 2$
by *algebra*
also have ... $\leq a \hat{} 2 + 2 * a \hat{} 2 * b + b \hat{} 2$
using *linear-le-pow* **by** *simp*
also have ... $= Suc\ (2 * b) * a \hat{} 2 + b \hat{} 2$
by *simp*
finally show *?thesis* **.**
qed
have *1*: $t < N$
using $t\ T'$ **by** *simp*
then have *15*: $t \leq N$
by *simp*
have *2*: $nlength\ t < N$
using *1 nlength-le-n dual-order.strict-trans2* **by** *blast*
have *25*: $nlength\ T \leq N$
using $T\ T'$ *nlength-le-n* **by** (*meson le-trans order-less-imp-le*)
have *27*: $nlength\ (t - 1) < N$
using *t*(*1*) *nlength-mono 2*
by (*metis diff-less dual-order.strict-trans2 less-numeral-extra*(*1*) *linorder-not-less order.asym*)
have *3*: $t * Z < N * Z$
using *1 Z-ge-1* **by** *simp*
then have *4*: $N + t * Z < Suc\ Z * N$
using *1* **by** *simp*
have *41*: $N + t * Z + Z \leq Suc\ Z * N$
proof $-$
have $N + t * Z + Z \leq N + (N - 1) * Z + Z$
using *1 N* **by** *auto*
then show *?thesis*
using *N*
by (*metis One-nat-def Suc-n-not-le-n ab-semigroup-add-class.add-ac*(*1*) *add.commute mult.commute mult-eq-if times-nat.simps*(*2*))
qed
have *42*: $(t + Z)\hat{}2 \leq (N + Z) \hat{} 2$
using *1* **by** *simp*
have *45*: $(N + Z) \hat{} 2 \leq 3*Z*N\hat{}2 + Z\hat{}2$
proof $-$
have $(N + Z) \hat{} 2 \leq Suc\ (2 * Z) * N\hat{}2 + Z\hat{}2$
using *sum-sq* **by** *simp*
also have ... $\leq 3*Z * N\hat{}2 + Z\hat{}2$
using *Z-ge-1* **by** *simp*
finally show *?thesis* **.**
qed
have *5*: $nlength\ (previous\ hp1\ t) < N$
using *previous-hp1-le 1* **by** (*meson dual-order.strict-trans2 nlength-le-n*)
then have *51*: $nlength\ (previous\ hp1\ t) \leq N$
by *simp*
have *6*: $nlength\ (N + t * Z) < Suc\ Z * N$
using *4 nlength-le-n* **by** (*meson le-trans linorder-not-le*)
have $nllength\ \sigma \leq 12 * N * Z\hat{}2 + 4 * Z * N$
proof $-$
have $nllength\ \sigma \leq 12 * T' * Z\hat{}2 + 4 * Z * N$
using *nllength-σ* **by** *simp*

also have ... ≤ 12 * N * Z^2 + 4 * Z * N
using T' by simp
finally show ?thesis .
qed
have 7: previous hp1 t ≤ N
using previous-hp1-le 15 by simp
have 65: (nlength (previous hp1 t) + nlength Z)^2 < (N + Z) ^ 2
proof −
have nlength (previous hp1 t) + nlength Z < N + Z
using 7 2 5 by (simp add: add-less-le-mono nlength-le-n)
then show ?thesis
by (simp add: power-strict-mono)
qed
have 66: N + previous hp1 t * Z + Z ≤ Suc Z * N
using 41 previous-hp1-le by (meson add-le-mono le-trans less-or-eq-imp-le mult-le-mono)
have 67: (nlength t + nlength Z)^2 ≤ 3 * Z * N^2 + Z^2
proof −
have nlength t + nlength Z < N + Z
using nlength-le-n 2 by (simp add: add-less-le-mono)
then have (nlength t + nlength Z)^2 < (N + Z) ^ 2
by (simp add: power-strict-mono)
then show ?thesis
using 45 by simp
qed
have nlength (previous hp1 t * Z) ≤ N * Z
using nlength-le-n previous-hp1-le 1 by (meson le-trans less-or-eq-imp-le mult-le-mono)
have 75: max (nlength N) (nlength (t * Z)) ≤ Suc Z * N
proof −
have max (nlength N) (nlength (t * Z)) ≤ nlength (max N (t * Z))
using max-nlength by simp
also have ... ≤ nlength (N + t * Z)
by (simp add: nlength-mono)
finally show ?thesis
using 6 by simp
qed
then have 78: max (nlength N) (nlength (previous hp1 t * Z)) ≤ Suc Z * N
using previous-hp1-le nlength-mono by (smt (verit, best) Groups.mult-ac(2) le-trans max-def mult-le-mono2)
have 79: nlength (N + t * Z + Z) ≤ Suc Z * N + Z
proof −
have N + t * Z + Z ≤ Suc Z * N + Z
using previous-le 15 by simp
then show ?thesis
using nlength-le-n le-trans by blast
qed
have 8: nllength [N + previous hp1 t * Z..<N + previous hp1 t * Z + Z] ≤ 2 * Z^2 * N + 2 * Z^2
proof −
have nllength [N + previous hp1 t * Z..<N + previous hp1 t * Z + Z] ≤
Suc (nlength (N + previous hp1 t * Z + Z)) * Z
using nllength-upt-le-len-mult-max by (metis diff-add-inverse)
moreover have nlength (N + previous hp1 t * Z + Z) ≤ Suc Z * N + Z
proof −
have N + previous hp1 t * Z + Z ≤ Suc Z * N + Z
using previous-le 15 7 by simp
then show ?thesis
using nlength-le-n le-trans by blast
qed
ultimately have nllength [N + previous hp1 t * Z..<N + previous hp1 t * Z + Z] ≤ Suc (Suc Z * N + Z)
* Z
by (meson Suc-le-mono le-trans less-or-eq-imp-le mult-le-mono)
also have ... = (Z^2 + Z) * Suc N
by (metis add.commute mult.commute mult.left-commute mult-Suc nat-arith.suc1 power2-eq-square)
also have ... ≤ (Z^2 + Z^2) * Suc N
by (meson add-le-cancel-left linear-le-pow mult-le-mono1 rel-simps(51))

also have ... = *2* ∗ *Z*ˆ*2* ∗ *Suc N*
  by *simp*
also have ... = *2* ∗ *Z*ˆ*2* ∗ *N* + *2* ∗ *Z*ˆ*2*
  by *simp*
finally show *?thesis* **.**
qed
have *84*: *Z* ∗ *Suc Z* ≤ *2* ∗ *Z*ˆ*2*
  by (*simp add*: *power2-eq-square*)
have *85*: *nlength* (*N* + *previous hp1 t* ∗ *Z*) ≤ *Suc Z* ∗ *N*
proof −
  have *nlength* (*N* + *previous hp1 t* ∗ *Z*) ≤ *nlength* (*N* + *t* ∗ *Z*)
    using *previous-hp1-le nlength-mono* **by** *simp*
  then show *?thesis*
    using *6* **by** *simp*
qed
have *9*: *Suc Z* ≤ *2*∗*Z*
  using *Z-ge-1* **by** *simp*
then have *91*: *Suc Z* ˆ *2* ≤ *4* ∗ *Z*ˆ*2*
  by (*metis mult-2-right numeral-Bit0 power2-eq-square power2-nat-le-eq-le power-mult-distrib*)
have *99*: *Z*ˆ*2* ≥ *81*
proof −
  have *Z* ∗ *Z* ≥ *9* ∗ *9*
    using *Z-ge-9 mult-le-mono* **by** *presburger*
  moreover have *9* ∗ *9* = (*81*::*nat*)
    by *simp*
  ultimately show *?thesis*
    by (*simp add*: *power2-eq-square*)
qed

have *part1*: *ttt8* ≤ *241* ∗ *Z*ˆ*2* + *266* ∗ *Z*ˆ*2* ∗ *N* ˆ *6*
proof −
  have *ttt8* ≤ *168* + *153* ∗ *N* ˆ *6* + *5* ∗ *t* + *26* ∗ (*t* + *Z*)^2 + *47* ∗ *Z* + *15* ∗ *Z* ∗ (*N* + *t* ∗ *Z*)
    using *ttt8-def T′* **by** *simp*
  also have ... ≤ *168* + *153* ∗ *N* ˆ *6* + *5* ∗ *N* + *26* ∗ (*t* + *Z*)^2 + *47* ∗ *Z* + *15* ∗ *Z* ∗ (*N* + *t* ∗ *Z*)
    using *15* **by** *simp*
  also have ... ≤ *168* + *153* ∗ *N* ˆ *6* + *5* ∗ *N* + *26* ∗ (*N* + *Z*)^2 + *47* ∗ *Z* + *15* ∗ *Z* ∗ (*N* + *t* ∗ *Z*)
    using *42* **by** *simp*
  also have ... ≤ *168* + *153* ∗ *N* ˆ *6* + *5* ∗ *N* + *26* ∗ (*3*∗*Z*∗*N*ˆ*2* + *Z*ˆ*2*) + *47* ∗ *Z* + *15* ∗ *Z* ∗ (*N* + *t* ∗ *Z*)
    using *45* **by** *simp*
  also have ... ≤ *168* + *153* ∗ *N* ˆ *6* + *5* ∗ *N* + *26* ∗ (*3*∗*Z*∗*N*ˆ*2* + *Z*ˆ*2*) + *47* ∗ *Z* + *15* ∗ *Z* ∗ *Suc Z* ∗ *N*
    using *4* **by** (*metis* (*mono-tags*, *lifting*) *add-left-mono less-or-eq-imp-le mult.assoc mult-le-mono2*)
  also have ... ≤ *168* + *153* ∗ *N* ˆ *6* + *5* ∗ *N* + *26* ∗ (*3*∗*Z*∗*N*ˆ*2* + *Z*ˆ*2*) + *47* ∗ *Z* + *30* ∗ *Z*ˆ*2* ∗ *N*
    using ‹*Z* ∗ *Suc Z* ≤ *2* ∗ *Z*ˆ*2*› **by** *simp*
  also have ... = *168* + *153* ∗ *N* ˆ *6* + *5* ∗ *N* + *78* ∗ *Z* ∗ *N*ˆ*2* + *26*∗*Z*ˆ*2* + *47* ∗ *Z* + *30* ∗ *Z*ˆ*2* ∗ *N*
    by *simp*
  also have ... ≤ *168* + *158* ∗ *N* ˆ *6* + *78* ∗ *Z* ∗ *N*ˆ*2* + *73* ∗ *Z*ˆ*2* + *30* ∗ *Z*ˆ*2* ∗ *N*
    using *linear-le-pow*[*of 6 N*] *linear-le-pow*[*of 2 Z*] **by** *simp*
  also have ... ≤ *168* + *158* ∗ *N* ˆ *6* + *78* ∗ *Z*ˆ*2* ∗ *N*ˆ*2* + *73* ∗ *Z*ˆ*2* + *30* ∗ *Z*ˆ*2* ∗ *N*ˆ*2*
    using *linear-le-pow*
   by (*metis add-le-mono add-le-mono1 le-square mult-le-mono1 mult-le-mono2 nat-add-left-cancel-le power2-eq-square*)
  also have ... = *168* + *158* ∗ *N* ˆ *6* + *108* ∗ *Z*ˆ*2* ∗ *N*ˆ*2* + *73* ∗ *Z*ˆ*2*
    by *simp*
  also have ... ≤ *168*∗*Z*ˆ*2* + *158* ∗ *N* ˆ *6* + *108* ∗ *Z*ˆ*2* ∗ *N*ˆ*2* + *73* ∗ *Z*ˆ*2*
    using *Z-ge-1* **by** *simp*
  also have ... ≤ *241* ∗ *Z*ˆ*2* + *158* ∗ *N* ˆ *6* + *108* ∗ *Z*ˆ*2* ∗ *N*ˆ*6*
    using *pow-mono′*[*of 2 6 N*] **by** *simp*
  also have ... ≤ *241* ∗ *Z*ˆ*2* + *158* ∗ *Z*ˆ*2* ∗ *N* ˆ *6* + *108* ∗ *Z*ˆ*2* ∗ *N*ˆ*6*
    using *Z-ge-1* **by** *simp*
  also have ... = *241* ∗ *Z*ˆ*2* + *266* ∗ *Z*ˆ*2* ∗ *N* ˆ *6*
    by *simp*
  finally show *?thesis* **.**
qed

have *part2*: $ttt10 - ttt8 \leq 63 * Z\hat{}2 + 226 * Z\hat{}2 * N\hat{}6$
proof −
  have $ttt10 - ttt8 = 37 + 26 * (nlength\ (previous\ hp1\ t) + nlength\ Z)^2 +$
    $3 * max\ (nlength\ N)\ (nlength\ (previous\ hp1\ t * Z)) +$
    $Suc\ Z * (43 + 9 * nlength\ (N + previous\ hp1\ t * Z + Z)) +$
    $4 * nllength\ [N + previous\ hp1\ t * Z..<N + previous\ hp1\ t * Z + Z] +$
    $2 * nlength\ (N + previous\ hp1\ t * Z)$
   using *ttt10-def ttt8-def* **by** *simp*
  also have $... \leq 37 + 26 * (nlength\ (previous\ hp1\ t) + nlength\ Z)^2 +$
    $3 * max\ (nlength\ N)\ (nlength\ (previous\ hp1\ t * Z)) +$
    $Suc\ Z * (43 + 9 * nlength\ (N + previous\ hp1\ t * Z + Z)) +$
    $4 * (2 * Z\hat{}2 * N + 2 * Z\hat{}2) + 2 * nlength\ (N + previous\ hp1\ t * Z)$
   using *8* **by** *simp*
  also have $... \leq 37 + 26 * (N + Z)^2 +$
    $3 * max\ (nlength\ N)\ (nlength\ (previous\ hp1\ t * Z)) +$
    $Suc\ Z * (43 + 9 * nlength\ (N + previous\ hp1\ t * Z + Z)) +$
    $4 * (2 * Z\hat{}2 * N + 2 * Z\hat{}2) + 2 * nlength\ (N + previous\ hp1\ t * Z)$
   using *65* **by** *linarith*
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * (Suc\ Z * N) +$
    $Suc\ Z * (43 + 9 * nlength\ (N + previous\ hp1\ t * Z + Z)) +$
    $8 * Z\hat{}2 * N + 8 * Z\hat{}2 + 2 * nlength\ (N + previous\ hp1\ t * Z)$
   using *78 45* **by** *auto*
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * (Suc\ Z * N) +$
    $Suc\ Z * (43 + 9 * nlength\ (N + previous\ hp1\ t * Z + Z)) +$
    $8 * Z\hat{}2 * N + 8 * Z\hat{}2 + 2 * Suc\ Z * N$
   using *85* **by** *linarith*
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * (Suc\ Z * N) +$
    $Suc\ Z * (43 + 9 * nlength\ (Suc\ Z * N)) +$
    $8 * Z\hat{}2 * N + 8 * Z\hat{}2 + 2 * Suc\ Z * N$
   using *66 nlength-mono add-le-mono le-refl mult-le-mono* **by** *presburger*
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * (Suc\ Z * N) +$
    $Suc\ Z * (43 + 9 * (Suc\ Z * N)) +$
    $8 * Z\hat{}2 * N + 8 * Z\hat{}2 + 2 * Suc\ Z * N$
   using *nlength-le-n add-le-mono le-refl mult-le-mono* **by** *presburger*
  also have $... = 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * (Suc\ Z * N) +$
    $43 * Suc\ Z + Suc\ Z * 9 * Suc\ Z * N +$
    $8 * Z\hat{}2 * N + 8 * Z\hat{}2 + 2 * Suc\ Z * N$
   by *algebra*
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * (Suc\ Z * N) +$
    $43 * 2 * Z + 2 * Z * 9 * Suc\ Z * N +$
    $8 * Z\hat{}2 * N + 8 * Z\hat{}2 + 2 * Suc\ Z * N$
   using *9* **by** (*simp add*: *add-le-mono*)
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * (Suc\ Z * N) +$
    $86 * Z + 36 * Z * Z * N + 8 * Z\hat{}2 * N + 8 * Z\hat{}2 + 2 * Suc\ Z * N$
   by *simp*
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * (Suc\ Z * N) +$
    $86 * Z + 44 * Z\hat{}2 * N + 8 * Z\hat{}2 + 2 * Suc\ Z * N$
   by (*simp add*: *power2-eq-square*)
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * (Suc\ Z * N) +$
    $86 * Z + 44 * Z\hat{}2 * N + 8 * Z\hat{}2 + 4 * Z * N$
   using *9* **by** *simp*
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * (Suc\ Z * N) +$
    $86 * Z + 48 * Z\hat{}2 * N + 8 * Z\hat{}2$
   using *linear-le-pow* **by** *simp*
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 3 * 2 * Z * N +$
    $86 * Z + 48 * Z\hat{}2 * N + 8 * Z\hat{}2$
   using *9* **by** *simp*
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 6 * Z * N +$
    $86 * Z\hat{}2 * N + 48 * Z\hat{}2 * N + 8 * Z\hat{}2$
   using *linear-le-pow*[*of 2 Z*] *N* **by** *simp* (*metis N le-trans mult-le-mono1 nat-mult-1*)
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 140 * Z\hat{}2 * N + 8 * Z\hat{}2$
   using *linear-le-pow*[*of 2 Z*] **by** *simp*
  also have $... \leq 37 + 26 * (3*Z*N\hat{}2 + Z\hat{}2) + 148 * Z\hat{}2 * N$

using *N* **by** *simp*
also have ... = *37* + *78* ∗ *Z* ∗ *N*^*2* + *26* ∗ *Z*^*2* + *148* ∗ *Z*^*2* ∗ *N*
by *simp*
also have ... ≤ *63* ∗ *Z*^*2* + *78* ∗ *Z* ∗ *N*^*2* + *148* ∗ *Z*^*2* ∗ *N*
using *Z-ge-1* **by** *simp*
also have ... ≤ *63* ∗ *Z*^*2* + *78* ∗ *Z*^*2* ∗ *N*^*2* + *148* ∗ *Z*^*2* ∗ *N*
using *linear-le-pow* **by** *simp*
also have ... ≤ *63* ∗ *Z*^*2* + *226* ∗ *Z*^*2* ∗ *N*^*2*
using *linear-le-pow* **by** *simp*
also have ... ≤ *63* ∗ *Z*^*2* + *226* ∗ *Z*^*2* ∗ *N*^*6*
using *pow-mono'*[*of 2 6 N*] **by** *simp*
finally show *?thesis* **.**
qed

have *10*: *nllength hp0* ≤ *N* ^ *2*
proof −
have ∀ *n*∈*set hp0*. *n* ≤ *T'*
using *hp0* **by** (*metis in-set-conv-nth*)
then have *nllength hp0* ≤ *Suc T'* ∗ *Suc T'*
using *nllength-le*[*of hp0 T'*] *len-hp0* **by** *simp*
also have ... ≤ *N* ∗ *N*
using *T' Suc-leI mult-le-mono* **by** *presburger*
also have ... = *N* ^ *2*
by *algebra*
finally show *?thesis* **.**
qed
have *11*: *nllength* [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] ≤ *2* ∗ *Z*^*2* ∗ *N* + *Z*
proof −
have *nllength* [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] ≤ *Suc* (*N* + *t* ∗ *Z* + *Z*) ∗ *Z*
using *nllength-upt-le*[*of N* + *t* ∗ *Z N* + *t* ∗ *Z* + *Z*] **by** *simp*
also have ... ≤ *Suc* (*Suc Z* ∗ *N*) ∗ *Z*
using *41* **by** *simp*
also have ... = (*Z*∗*Z* + *Z*)∗*N* + *Z*
by (*metis add.commute mult.commute mult.left-commute mult-Suc*)
also have ... ≤ *2* ∗ *Z*^*2* ∗ *N* + *Z*
using *linear-le-pow*[*of 2 Z*] **by** (*simp add*: *power2-eq-square*)
finally show *?thesis* **.**
qed
have *12*: *nlength* (*hp0* ! *t*) + *nlength G* ≤ *N* + *Z*
proof −
have *nlength* (*hp0* ! *t*) + *nlength G* ≤ *hp0* ! *t* + *G*
using *nlength-le-n* **by** (*simp add*: *add-mono*)
also have ... ≤ *T'* + *Z*
using *Z* **by** (*simp add*: *add-le-mono hp0 le-imp-less-Suc len-hp0 t*(*2*))
also have ... ≤ *N* + *Z*
using *T'* **by** *simp*
finally show *?thesis* **.**
qed

have *part3*: *ttt20* − *ttt10* ≤ *120* ∗ *Z*^*2* + *206* ∗ *Z*^*2* ∗ *N* ^ *4*
proof −
have *ttt20* − *ttt10* = *80* + *2* ∗ *nlength* (*t* − *1*) + *26* ∗ (*nlength t* + *nlength Z*)^2 +
3 ∗ *max* (*nlength N*) (*nlength* (*t* ∗ *Z*)) + *Suc Z* ∗ (*43* + *9* ∗ *nlength* (*N* + *t* ∗ *Z* + *Z*)) +
4 ∗ *nllength* [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] + *2* ∗ *nlength* (*previous hp1 t*) +
21 ∗ (*nllength hp0*)^2 + *2* ∗ *nlength* (*N* + *t* ∗ *Z*) + *26* ∗ (*nlength* (*hp0* ! *t*) + *nlength G*)^2
using *ttt20-def ttt10-def* **by** *simp*
also have ... ≤ *80* + *2* ∗ *N* + *26* ∗ (*3* ∗ *Z* ∗ *N*^*2* + *Z*^*2*) +
3 ∗ (*Suc Z* ∗ *N*) + *Suc Z* ∗ (*43* + *9* ∗ *nlength* (*N* + *t* ∗ *Z* + *Z*)) +
4 ∗ *nllength* [*N* + *t* ∗ *Z*..<*N* + *t* ∗ *Z* + *Z*] + *2* ∗ *nlength* (*previous hp1 t*) +
21 ∗ (*nllength hp0*)^2 + *2* ∗ *nlength* (*N* + *t* ∗ *Z*) + *26* ∗ (*nlength* (*hp0* ! *t*) + *nlength G*)^2
using *27 67 75* **by** *auto*
also have ... ≤ *80* + *2* ∗ *N* + *26* ∗ (*3* ∗ *Z* ∗ *N*^*2* + *Z*^*2*) +
3 ∗ (*Suc Z* ∗ *N*) + *Suc Z* ∗ (*43* + *9* ∗ (*Suc Z* ∗ *N* + *Z*)) +

$4 * nllength\ [N + t * Z..<N + t * Z + Z] + 2 * nlength\ (previous\ hp1\ t) +$
$21 * (nllength\ hp0)^2 + 2 * nlength\ (N + t * Z) + 26 * (nlength\ (hp0\ !\ t) + nlength\ G)^2$
using *79 add-le-mono le-refl mult-le-mono* **by** *presburger*
also have ... $\leq 80 + 2 * N + 26 * (3 * Z * N^\frown 2 + Z^\frown 2) +$
$3 * (Suc\ Z * N) + Suc\ Z * (43 + 9 * (Suc\ Z * N + Z)) +$
$4 * nllength\ [N + t * Z..<N + t * Z + Z] + 2 * N +$
$21 * (nllength\ hp0)^2 + 2 * nlength\ (N + t * Z) + 26 * (N + Z)^2$
using *51 12*
by (*metis add-le-cancel-right add-le-mono nat-add-left-cancel-le nat-mult-le-cancel-disj power2-nat-le-eq-le*)
also have ... $\leq 80 + 4 * N + 52 * (3 * Z * N^\frown 2 + Z^\frown 2) + 3 * (Suc\ Z * N) + Suc\ Z * (43 + 9 * (Suc\ Z * N + Z)) +$
$4 * nllength\ [N + t * Z..<N + t * Z + Z] + 21 * (nllength\ hp0)^2 + 2 * nlength\ (N + t * Z)$
using *45* **by** *simp*
also have ... $\leq 80 + 4 * N + 52 * (3 * Z * N^\frown 2 + Z^\frown 2) + 3 * (Suc\ Z * N) + Suc\ Z * (43 + 9 * (Suc\ Z * N + Z)) +$
$4 * nllength\ [N + t * Z..<N + t * Z + Z] + 21 * (nllength\ hp0)^2 + 2 * Suc\ Z * N$
using *6* **by** *linarith*
also have ... $\leq 80 + 4 * N + 52 * (3 * Z * N^\frown 2 + Z^\frown 2) + 3 * (Suc\ Z * N) + Suc\ Z * (43 + 9 * (Suc\ Z * N + Z)) +$
$4 * (2 * Z^\frown 2 * N + Z) + 21 * (N^\frown 2)^2 + 2 * Suc\ Z * N$
using *11 10 add-le-mono le-refl mult-le-mono2 power2-nat-le-eq-le* **by** *presburger*
also have ... $= 80 + 4 * N + 156 * Z * N^\frown 2 + 52 * Z^\frown 2 + 3 * (Suc\ Z * N) + Suc\ Z * (43 + 9 * (Suc\ Z * N + Z)) +$
$8 * Z^\frown 2 * N + 4 * Z + 21 * N\ ^\frown\ 4 + 2 * Suc\ Z * N$
by *simp*
also have ... $= 80 + 156 * Z * N^\frown 2 + 52 * Z^\frown 2 + Suc\ Z * 43 + Suc\ Z * 9 * Suc\ Z * N + Suc\ Z * 9 * Z +$
$(4 + 5 * Suc\ Z + 8 * Z^\frown 2) * N + 4 * Z + 21 * N\ ^\frown\ 4$
by *algebra*
also have ... $= 123 + 156 * Z * N^\frown 2 + 52 * Z^\frown 2 + 47 * Z + 9 * Suc\ Z * Suc\ Z * N + Suc\ Z * 9 * Z +$
$(9 + 5 * Z + 8 * Z^\frown 2) * N + 21 * N\ ^\frown\ 4$
by *simp*
also have ... $= 123 + 156 * Z * N^\frown 2 + 52 * Z^\frown 2 + 47 * Z + 9 * Z * Suc\ Z +$
$(9 * Suc\ Z^\frown 2 + 9 + 5 * Z + 8 * Z^\frown 2) * N + 21 * N\ ^\frown\ 4$
by *algebra*
also have ... $\leq 123 + 156 * Z * N^\frown 2 + 52 * Z^\frown 2 + 47 * Z + 9 * Z * Suc\ Z +$
$(44 * Z^\frown 2 + 9 + 5 * Z) * N + 21 * N\ ^\frown\ 4$
using *91* **by** *simp*
also have ... $\leq 123 + 156 * Z * N^\frown 2 + 70 * Z^\frown 2 + 47 * Z + (44 * Z^\frown 2 + 9 + 5 * Z) * N + 21 * N\ ^\frown\ 4$
using *84* **by** *simp*
also have ... $\leq 123 + 156 * Z * N^\frown 2 + 70 * Z^\frown 2 + 47 * Z^\frown 2 + (44 * Z^\frown 2 + 9 + 5 * Z^\frown 2) * N + 21 * N\ ^\frown\ 4$
using *linear-le-pow*[*of 2 Z*] *add-le-mono le-refl mult-le-mono power2-nat-le-imp-le*
by *presburger*
also have ... $\leq 123 + 156 * Z * N^\frown 2 + 117 * Z^\frown 2 + (49 * Z^\frown 2 + 9) * N\ ^\frown\ 4 + 21 * N\ ^\frown\ 4$
using *linear-le-pow*[*of 4 N*] **by** *simp*
also have ... $\leq 123 + 156 * Z * N^\frown 4 + 117 * Z^\frown 2 + (49 * Z^\frown 2 + 9) * N\ ^\frown\ 4 + 21 * N\ ^\frown\ 4$
using *pow-mono'*[*of 2 4 N*] **by** *simp*
also have ... $= 123 + 117 * Z^\frown 2 + (156 * Z + 49 * Z^\frown 2 + 30) * N\ ^\frown\ 4$
by *algebra*
also have ... $\leq 123 + 117 * Z^\frown 2 + (205 * Z^\frown 2 + 30) * N\ ^\frown\ 4$
using *linear-le-pow*[*of 2 Z*] **by** *simp*
also have ... $\leq 120 * Z^\frown 2 + (205 * Z^\frown 2 + 30) * N\ ^\frown\ 4$
using *99* **by** *simp*
also have ... $\leq 120 * Z^\frown 2 + 206 * Z^\frown 2 * N\ ^\frown\ 4$
using *99* **by** *simp*
finally show *?thesis* **.**
qed

have *12*: $hp0\ !\ t * G + G \leq Z * N$
proof −
have $hp0\ !\ t * G + G \leq T' * G + G$
using *hp0 t*(*2*) *len-hp0* **by** *simp*

also have ... ≤ (N − 1) ∗ G + G
using T′ by auto
also have ... = N ∗ G
using T′ by (metis Suc-diff-1 add.commute less-nat-zero-code mult-Suc not-gr-zero)
also have ... ≤ Z ∗ N
using Z by simp
finally show ?thesis .
qed
have 13: Suc G ∗ (43 + 9 ∗ nlength (hp0 ! t ∗ G + G)) ≤ 43 ∗ Z + 9 ∗ Z^2 ∗ N
proof −
have Suc G ∗ (43 + 9 ∗ nlength (hp0 ! t ∗ G + G)) ≤ Suc G ∗ (43 + 9 ∗ (hp0 ! t ∗ G + G))
using nlength-le-n add-le-mono le-refl mult-le-mono by presburger
also have ... ≤ Suc G ∗ (43 + 9 ∗ (Z ∗ N))
using 12 add-le-mono less-or-eq-imp-le mult-le-mono by presburger
also have ... ≤ Z ∗ (43 + 9 ∗ (Z ∗ N))
using G Z by simp
also have ... = 43 ∗ Z + 9 ∗ N ∗ Z ∗ Z
by algebra
also have ... = 43 ∗ Z + 9 ∗ Z^2 ∗ N
by algebra
finally show ?thesis .
qed
have 14: nllength [hp0 ! t ∗ G..<hp0 ! t ∗ G + G] ≤ Z^2 ∗ N
proof −
have nllength [hp0 ! t ∗ G..<hp0 ! t ∗ G + G] ≤ (hp0 ! t ∗ G + G) ∗ Suc G
using nllength-upt-le[of hp0 ! t ∗ G hp0 ! t ∗ G + G] by simp
also have ... ≤ (hp0 ! t ∗ G + G) ∗ Z
using Z G by simp
also have ... ≤ N ∗ Z ∗ Z
using 12 by (simp add: mult.commute)
also have ... = Z^2 ∗ N
by algebra
finally show ?thesis .
qed
have 15: nlength (hp0 ! t) ≤ N
using T′ hp0 t(2) len-hp0 nlength-le-n by (metis le-imp-less-Suc le-trans less-or-eq-imp-le)
have 16: nlength (hp0 ! t ∗ G) ≤ Z ∗ N
proof −
have nlength (hp0 ! t ∗ G) ≤ hp0 ! t ∗ G
using nlength-le-n by simp
also have ... ≤ T′ ∗ G
using Z T′ hp0 t(2) len-hp0 by simp
also have ... ≤ Z ∗ N
using Z T′ by simp
finally show ?thesis .
qed
have 17: (12 ∗ T′ ∗ Z^2 + 4 ∗ Z ∗ N) ^ 2 ≤ 256 ∗ Z^4 ∗ N^2
proof −
have (12 ∗ T′ ∗ Z^2 + 4 ∗ Z ∗ N) ^ 2 ≤ (12 ∗ N ∗ Z^2 + 4 ∗ Z ∗ N)^2
using T′ by simp
also have ... ≤ (12 ∗ N ∗ Z^2 + 4 ∗ Z^2 ∗ N)^2
using linear-le-pow[of 2 Z] add-le-mono le-refl mult-le-mono power2-nat-le-eq-le power2-nat-le-imp-le
by presburger
also have ... = 256 ∗ Z^4 ∗ N^2
by algebra
finally show ?thesis .
qed
have 18: 108 ∗ (24 ∗ Z^2 ∗ 2 ^ (4 ∗ Z))^2 ∗ (3 + (12 ∗ T′ ∗ Z^2 + 4 ∗ Z ∗ N)^2) ≤ 16111872 ∗ 2^(16∗Z) ∗ N^2
proof −
have 108 ∗ (24 ∗ Z^2 ∗ 2 ^ (4 ∗ Z))^2 = 62208 ∗ (Z^2 ∗ 2 ^ (4∗Z))^2
by algebra
also have ... = 62208 ∗ Z^(2∗2) ∗ 2 ^ (2∗(4∗Z))

by (*metis* (*no-types*, *lifting*) *mult.assoc power-even-eq power-mult-distrib*)
also have ... $= 62208 * Z\hat{}4 * 2 \hat{} (8*Z)$
by *simp*
finally have $*$: $108 * (24 * Z^2 * 2 \hat{} (4 * Z))^2 = 62208 * Z\hat{}4 * 2 \hat{} (8*Z)$ **.**
have $3 + (12 * T' * Z^2 + 4 * Z * N)^2 \leq 3 + 256 * Z\hat{}4 * N\hat{}2$
using *17* **by** *simp*
moreover have $Z\hat{}4 * N\hat{}2 \geq 1$
using *Z-ge-1 N* **by** *simp*
ultimately have $3 + (12 * T' * Z^2 + 4 * Z * N)^2 \leq 259 * Z\hat{}4 * N\hat{}2$
by *linarith*
then have $108 * (24 * Z^2 * 2 \hat{} (4 * Z))^2 * (3 + (12 * T' * Z^2 + 4 * Z * N)^2) \leq$
$16111872 * Z\hat{}4 * 2 \hat{} (8*Z) * Z\hat{}4 * N\hat{}2$
using $*$ **by** *simp*
also have ... $= 16111872 * Z\hat{}8 * 2 \hat{} (8*Z) * N\hat{}2$
by *simp*
also have ... $\leq 16111872 * 2\hat{}(8*Z) * 2 \hat{} (8*Z) * N\hat{}2$
using *pow8-le-2pow8* **by** *simp*
also have ... $= 16111872 * 2\hat{}(8*Z+8*Z) * N\hat{}2$
by (*metis* (*no-types*, *lifting*) *mult.assoc power-add*)
also have ... $= 16111872 * 2\hat{}(16*Z) * N\hat{}2$
by *simp*
finally show *?thesis* **.**
qed
have *19*: *nllength* $\sigma \leq 16 * Z\hat{}2 * N$
proof $-$
have *nllength* $\sigma \leq 12 * T' * Z\hat{}2 + 4 * Z * N$
using *nllength-σ* **by** *simp*
also have ... $\leq 12 * N * Z\hat{}2 + 4 * Z * N$
using T' **by** *simp*
also have ... $\leq 12 * Z\hat{}2 * N + 4 * Z\hat{}2 * N$
using *linear-le-pow*[*of 2 Z*] **by** *simp*
also have ... $\leq 16 * Z\hat{}2 * N$
by *simp*
finally show *?thesis* **.**
qed

have *part4*: *ttt27* $-$ *ttt20* $\leq 50 * Z + 16111936 * 2\hat{}(16*Z) * N\hat{}2$
proof $-$
have *ttt27* $-$ *ttt20* $= 55 + Suc\ G * (43 + 9 * nlength\ (hp0\ !\ t * G + G)) +$
$4 * nllength\ [hp0\ !\ t * G..<hp0\ !\ t * G + G] +$
$2 * nlength\ (hp0\ !\ t * G) + 2 * nlength\ (hp0\ !\ t) + 3 * nllength\ \sigma +$
$108 * (24 * Z^2 * 2 \hat{} (4 * Z))^2 * (3 + (12 * T' * Z^2 + 4 * Z * N)^2)$
using *ttt27-def ttt20-def* **by** *linarith*
also have ... $\leq 55 + Suc\ G * (43 + 9 * nlength\ (hp0\ !\ t * G + G)) +$
$4 * nllength\ [hp0\ !\ t * G..<hp0\ !\ t * G + G] +$
$2 * nlength\ (hp0\ !\ t * G) + 2 * nlength\ (hp0\ !\ t) + 3 * (16 * Z\hat{}2 * N) +$
$108 * (24 * Z^2 * 2 \hat{} (4 * Z))^2 * (3 + (12 * T' * Z^2 + 4 * Z * N)^2)$
using *19* **by** (*simp add*: *mult.commute*)
also have ... $\leq 55 + Suc\ G * (43 + 9 * nlength\ (hp0\ !\ t * G + G)) +$
$2 * Z * N + 2 * N + 52 * Z\hat{}2 * N + 16111872 * 2\hat{}(16*Z) * N\hat{}2$
using *14 15 16 18* **by** *linarith*
also have ... $\leq 55 + 43 * Z + 9 * Z\hat{}2 * N + 2 * Z * N + 2 * N + 52 * Z\hat{}2 * N + 16111872 * 2\hat{}(16*Z) * N\hat{}2$
using *13* **by** *linarith*
also have ... $= 55 + 43 * Z + 2 * Z * N + 2 * N + 61 * Z\hat{}2 * N + 16111872 * 2\hat{}(16*Z) * N\hat{}2$
by *simp*
also have ... $\leq 50 * Z + 2 * Z * N + 2 * N + 61 * Z\hat{}2 * N + 16111872 * 2\hat{}(16*Z) * N\hat{}2$
using *Z-ge-9* **by** *simp*
also have ... $\leq 50 * Z + 3 * Z * N + 61 * Z\hat{}2 * N + 16111872 * 2\hat{}(16*Z) * N\hat{}2$
using *Z-ge-9* **by** *simp*
also have ... $\leq 50 * Z + 64 * Z\hat{}2 * N + 16111872 * 2\hat{}(16*Z) * N\hat{}2$
using *linear-le-pow*[*of 2 Z*] **by** *simp*
also have ... $\leq 50 * Z + 64 * Z\hat{}2 * N\hat{}2 + 16111872 * 2\hat{}(16*Z) * N\hat{}2$

using *linear-le-pow*[*of 2 N*] **by** *simp*
also have ... $\leq$ *50* $*$ *Z* + *64* $*$ *2^(2*Z)* $*$ *N^2* + *16111872* $*$ *2^(16*Z)* $*$ *N^2*
using *pow2-le-2pow2* **by** *simp*
also have ... $\leq$ *50* $*$ *Z* + *64* $*$ *2^(16*Z)* $*$ *N^2* + *16111872* $*$ *2^(16*Z)* $*$ *N^2*
by *simp*
also have ... = *50* $*$ *Z* + *16111936* $*$ *2^(16*Z)* $*$ *N^2*
by *simp*
finally show *?thesis* **.**
qed

have *part5*: *24* + *4* $*$ (*Suc* (*nllength* σ) $*$ *24* $*$ *Z* ^ *2* $*$ *2* ^ (*4*Z*)) + *2* $*$ *nlength t* + *2* $*$ *nlength T* $\leq$
24 + *1633* $*$ *2^(8*Z)* $*$ *N*
proof −
have *24* + *4* $*$ (*Suc* (*nllength* σ) $*$ *24* $*$ *Z* ^ *2* $*$ *2* ^ (*4*Z*)) + *2* $*$ *nlength t* + *2* $*$ *nlength T* $\leq$
24 + *4* $*$ (*Suc* (*nllength* σ) $*$ *24* $*$ *Z* ^ *2* $*$ *2* ^ (*4*Z*)) + *4* $*$ *N*
using *25 2* **by** *simp*
also have ... $\leq$ *24* + *4* $*$ (*Suc* (*16* $*$ *Z^2* $*$ *N*) $*$ *24* $*$ *Z* ^ *2* $*$ *2* ^ (*4*Z*)) + *4* $*$ *N*
using *19* **by** (*simp add*: *mult.commute*)
also have ... $\leq$ *24* + *1632* $*$ *Z^2* $*$ *N* $*$ *Z* ^ *2* $*$ *2* ^ (*4*Z*) + *4* $*$ *N*
using *Z N* **by** *simp*
also have ... = *24* + *1632* $*$ *Z^4* $*$ *2* ^ (*4*Z*) $*$ *N* + *4* $*$ *N*
by *simp*
also have ... $\leq$ *24* + *1632* $*$ *2^(4*Z)* $*$ *2* ^ (*4*Z*) $*$ *N* + *4* $*$ *N*
using *pow4-le-2pow4* **by** *simp*
also have ... = *24* + *1632* $*$ *2^(8*Z)* $*$ *N* + *4* $*$ *N*
by (*metis* (*no-types*, *lifting*) *ab-semigroup-mult-class.mult-ac*(*1*) *add-mult-distrib numeral-Bit0 power-add*)
also have ... $\leq$ *24* + *1632* $*$ *2^(8*Z)* $*$ *N* + *2^(8*Z)* $*$ *N*
proof −
have (*4*::*nat*) $\leq$ *2* ^ *8*
by *simp*
also have ... $\leq$ *2* ^ (*8*Z*)
using *Z-ge-1* **by** (*metis nat-mult-1-right nat-mult-le-cancel-disj one-le-numeral power-increasing*)
finally have (*4*::*nat*) $\leq$ *2* ^ (*8*Z*) **.**
then show *?thesis*
by *simp*
qed
also have ... $\leq$ *24* + *1633* $*$ *2^(8*Z)* $*$ *N*
by *simp*
finally show *?thesis* **.**
qed

show *ttt27* + *24* + *4* $*$ (*Suc* (*nllength* σ) $*$ *24* $*$ *Z* ^ *2* $*$ *2* ^ (*4*Z*)) + *2* $*$ *nlength t* + *2* $*$ *nlength T* $\leq$
16114765 $*$ *2^(16*Z)* $*$ *N^6*
proof −
have *ttt27* $\leq$ *ttt20* + *50* $*$ *Z* + *16111936* $*$ *2^(16*Z)* $*$ *N^2*
using *part4 ttt27-def* **by** *simp*
also have ... $\leq$ *ttt10* + *120* $*$ *Z^2* + *206* $*$ *Z^2* $*$ *N* ^ *4* + *50* $*$ *Z* + *16111936* $*$ *2^(16*Z)* $*$ *N^2*
using *part3 ttt20-def* **by** *simp*
also have ... $\leq$ *ttt8* + *63* $*$ *Z^2* + *226* $*$ *Z^2* $*$ *N^6* + *120* $*$ *Z^2* + *206* $*$ *Z^2* $*$ *N* ^ *4* + *50* $*$ *Z* +
16111936 $*$ *2^(16*Z)* $*$ *N^2*
using *part2 ttt10-def* **by** *simp*
also have ... $\leq$ *241* $*$ *Z^2* + *266* $*$ *Z^2* $*$ *N* ^ *6* + *63* $*$ *Z^2* + *226* $*$ *Z^2* $*$ *N^6* + *120* $*$ *Z^2* +
206 $*$ *Z^2* $*$ *N* ^ *4* + *50* $*$ *Z* + *16111936* $*$ *2^(16*Z)* $*$ *N^2*
using *part1* **by** *simp*
also have ... = *424* $*$ *Z^2* + *492* $*$ *Z^2* $*$ *N* ^ *6* + *206* $*$ *Z^2* $*$ *N* ^ *4* + *50* $*$ *Z* + *16111936* $*$ *2^(16*Z)*
$*$ *N^2*
by *simp*
also have ... $\leq$ *474* $*$ *Z^2* + *492* $*$ *Z^2* $*$ *N* ^ *6* + *206* $*$ *Z^2* $*$ *N* ^ *4* + *16111936* $*$ *2^(16*Z)* $*$ *N^2*
using *linear-le-pow*[*of 2 Z*] **by** *simp*
also have ... $\leq$ *474* $*$ *Z^2* + *698* $*$ *Z^2* $*$ *N* ^ *6* + *16111936* $*$ *2^(16*Z)* $*$ *N^2*
using *pow-mono'*[*of 4 6 N*] **by** *simp*
also have ... $\leq$ *474* $*$ *Z^2* + *698* $*$ *Z^2* $*$ *N* ^ *6* + *16111936* $*$ *2^(16*Z)* $*$ *N^6*
using *pow-mono'*[*of 2 6 N*] **by** *simp*

also have ... $\leq$ *474 ∗ Z^2 + 698 ∗ 2^(16∗Z) ∗ N ^ 6 + 16111936 ∗ 2^(16∗Z) ∗ N^6*
using *Z-sq-le* **by** *simp*
also have ... = *474 ∗ Z^2 + 16112634 ∗ 2^(16∗Z) ∗ N^6*
by *simp*
also have ... $\leq$ *474 ∗ 2^(16∗Z) + 16112634 ∗ 2^(16∗Z) ∗ N^6*
using *Z-sq-le* **by** *simp*
also have ... $\leq$ *16113108 ∗ 2^(16∗Z) ∗ N^6*
using *N* **by** *simp*
finally have *ttt27* $\leq$ *16113108 ∗ 2^(16∗Z) ∗ N^6* **.**
then have *ttt27 + 24 + 4 ∗ (Suc (nllength σ) ∗ 24 ∗ Z ^ 2 ∗ 2 ^ (4∗Z)) + 2 ∗ nlength t + 2 ∗ nlength T*
$\leq$
16113108 ∗ 2^(16∗Z) ∗ N^6 + 24 + 1633 ∗ 2^(8∗Z) ∗ N
using *part5* **by** *simp*
also have ... $\leq$ *16113108 ∗ 2^(16∗Z) ∗ N^6 + 24 + 1633 ∗ 2^(16∗Z) ∗ N*
by *simp*
also have ... $\leq$ *16113108 ∗ 2^(16∗Z) ∗ N^6 + 24 + 1633 ∗ 2^(16∗Z) ∗ N^6*
using *linear-le-pow[of 6 N]* **by** *simp*
also have ... = *24 + 16114741 ∗ 2^(16∗Z) ∗ N^6*
by *simp*
also have ... $\leq$ *24 ∗ 2^(16∗Z) + 16114741 ∗ 2^(16∗Z) ∗ N^6*
using *Z-sq-le* **by** *simp*
also have ... $\leq$ *16114765 ∗ 2^(16∗Z) ∗ N^6*
using *N* **by** *simp*
finally show *?thesis* **.**
qed
qed

lemma *tm30′*:
assumes *ttt = 16114765 ∗ 2^(16∗Z) ∗ N^6*
shows *transforms tm30 tps0 ttt tps30*
using *tm30 time-bound transforms-monotone assms* **by** *simp*

end

end

lemma *transforms-tm-chi*:
fixes *j1 j2 j3 :: tapeidx*
fixes *tps tps′ :: tape list* **and** *k G N Z T′ T t :: nat* **and** *hp0 hp1 :: nat list* **and** *ψ ψ′ :: formula*
fixes *nss :: nat list list*
assumes *length tps = k*
and *1 < j1 j1 < j2 j2 < j3 j3 + 17 < k*
and *G ≥ 3*
and *Z = 3 ∗ G*
and *N ≥ 1*
and *length hp0 = Suc T′*
and *∀ i<length hp0. hp0 ! i ≤ T′*
and *length hp1 = Suc T′*
and *∀ i<length hp1. hp1 ! i ≤ T′*
and *0 < t t ≤ T′*
and *0 < T T ≤ T′*
and *T′ < N*
and *variables ψ ⊆ {..<3∗Z+G} fsize ψ ≤ (3∗Z+G) ∗ 2 ^ (3∗Z+G) length ψ ≤ 2 ^ (3∗Z+G)*
and *variables ψ′ ⊆ {..<2∗Z+G} fsize ψ′ ≤ (2∗Z+G) ∗ 2 ^ (2∗Z+G) length ψ′ ≤ 2 ^ (2∗Z+G)*
assumes
tps ! 1 = nlltape nss
tps ! j1 = $(\lfloor hp0 \rfloor_{NL}, 1)$
tps ! j2 = $(\lfloor hp1 \rfloor_{NL}, 1)$
tps ! j3 = $(\lfloor N \rfloor_{N}, 1)$
tps ! (j3 + 1) = $(\lfloor G \rfloor_{N}, 1)$
tps ! (j3 + 2) = $(\lfloor Z \rfloor_{N}, 1)$
tps ! (j3 + 3) = $(\lfloor T \rfloor_{N}, 1)$
tps ! (j3 + 4) = $(\lfloor formula\text{-}n\ \psi \rfloor_{NLL}, 1)$

```
    tps ! (j3 + 5) = (⌊formula-n ψ'⌋NLL, 1)
    tps ! (j3 + 6) = (⌊t⌋N, 1)
    ⋀i. 6 < i ⟹ i < 17 ⟹ tps ! (j3 + i) = (⌊[]⌋, 1)
  assumes tps' = tps
        [1 := nlltape (nss @ formula-n (relabel
          ([N + (t − 1) ∗ Z..<N + (t − 1) ∗ Z + Z] @
            (if previous hp1 t ≠ t then [N + previous hp1 t ∗ Z..<N + previous hp1 t ∗ Z + Z] else []) @
            [N + t ∗ Z..<N + t ∗ Z + Z] @
            [hp0 ! t ∗ G..<hp0 ! t ∗ G + G])
          (if previous hp1 t = t then ψ' else ψ))),
         j3 + 6 := (⌊Suc t⌋N, 1),
         j3 + 3 := (⌊T − 1⌋N, 1)]
  assumes ttt = 16114765 ∗ 2 ^ (16∗Z) ∗ N^6
  shows transforms (tm-chi j1 j2 j3) tps ttt tps'
proof −
  interpret loc: turing-machine-chi j1 j2 j3 .
  show ?thesis
    using loc.tm30'[OF assms(1−34)] loc.tps30-def[OF assms(1−34)] assms(35,36) loc.tm30-eq-tm-chi
    by simp
qed
```

A Turing machine for Φ_9 proper

The formula Φ_9 is a conjunction of formulas χ_t. The TM *tm-chi* decreases the number on tape $j_3 + 3$. If this tape is initialized with T', then a while loop with *tm-chi* as its body will generate Φ_9. The next TM is such a machine:

```
definition tm-PHI9 :: tapeidx ⇒ tapeidx ⇒ tapeidx ⇒ machine where
  tm-PHI9 j1 j2 j3 ≡ WHILE [] ; λrs. rs ! (j3 + 3) ≠ □ DO tm-chi j1 j2 j3 DONE

lemma tm-PHI9-tm:
  assumes 0 < j1 and j1 < j2 and j2 < j3 and j3 + 17 < k and G ≥ 6
  shows turing-machine k G (tm-PHI9 j1 j2 j3)
  unfolding tm-PHI9-def
  using tm-chi-tm turing-machine-loop-turing-machine Nil-tm assms by simp

lemma map-nth:
  fixes zs ys f n i
  assumes zs = map f [0..<n] and i < length zs
  shows zs ! i = f i
  using assms by simp

lemma concat-formula-n:
  concat (map (λt. formula-n (φ t)) [0..<n]) = formula-n (concat (map (λt. φ t) [0..<n]))
  using formula-n-def by (induction n) simp-all

lemma upt-append-upt:
  assumes a ≤ b and b ≤ c
  shows [a..<b] @ [b..<c] = [a..<c]
proof (rule nth-equalityI)
  show length ([a..<b] @ [b..<c]) = length [a..<c]
    using assms by simp
  show ([a..<b] @ [b..<c]) ! i = [a..<c] ! i if i < length ([a..<b] @ [b..<c]) for i
    using assms that nth-append[of [a..<b] [b..<c] i] by (cases i < b − a) simp-all
qed
```

The semantics of the TM *tm-PHI9* can be proved inside the locale *reduction-sat-x* because it is a fairly simple TM.

```
context reduction-sat-x
begin
```

The TM *tm-chi* is the first TM whose semantics we transfer into the locale *reduction-sat-x*.

```
lemma tm-chi:
```

fixes *tps0* :: *tape list* **and** *k t′ t* :: *nat* **and** *j1 j2 j3* :: *tapeidx*
fixes *nss* :: *nat list list*
assumes *jk*: $length\ tps0 = k\ 1 < j1\ j1 < j2\ j2 < j3\ j3 + 17 < k$
and *t*: $0 < t\ t \leq T'$
and *T*: $0 < t'\ t' \leq T'$
assumes $hp0 = map\ (\lambda t.\ exc\ (zeros\ m)\ t <\#> 0)\ [0..<Suc\ T']$
assumes $hp1 = map\ (\lambda t.\ exc\ (zeros\ m)\ t <\#> 1)\ [0..<Suc\ T']$
assumes *tps0*:
$tps0\ !\ 1 = nlltape\ nss$
$tps0\ !\ j1 = (\lfloor hp0 \rfloor_{NL},\ 1)$
$tps0\ !\ j2 = (\lfloor hp1 \rfloor_{NL},\ 1)$
$tps0\ !\ j3 = (\lfloor N \rfloor_{N},\ 1)$
$tps0\ !\ (j3 + 1) = (\lfloor H \rfloor_{N},\ 1)$
$tps0\ !\ (j3 + 2) = (\lfloor Z \rfloor_{N},\ 1)$
$tps0\ !\ (j3 + 3) = (\lfloor t' \rfloor_{N},\ 1)$
$tps0\ !\ (j3 + 4) = (\lfloor formula\text{-}n\ \psi \rfloor_{NLL},\ 1)$
$tps0\ !\ (j3 + 5) = (\lfloor formula\text{-}n\ \psi' \rfloor_{NLL},\ 1)$
$tps0\ !\ (j3 + 6) = (\lfloor t \rfloor_{N},\ 1)$
$\bigwedge i.\ 6 < i \Longrightarrow i < 17 \Longrightarrow tps0\ !\ (j3 + i) = (\lfloor [] \rfloor,\ 1)$
assumes $tps' = tps0$
$[1 := nlltape\ (nss\ @\ formula\text{-}n\ (\chi\ t)),$
$j3 + 6 := (\lfloor Suc\ t \rfloor_{N},\ 1),$
$j3 + 3 := (\lfloor t' - 1 \rfloor_{N},\ 1)]$
assumes $ttt = 16114765 * 2\hat{}(16*Z) * N \hat{} 6$
shows *transforms* (*tm-chi j1 j2 j3*) *tps0 ttt tps′*
proof −
interpret *loc*: *turing-machine-chi j1 j2 j3* **.**

have *G*: $H \geq 3$
using *H-gr-2* **by** *simp*
then have *N*: $N \geq 1$
using *N-eq* **by** *simp*
have *Z*: $Z = 3 * H$
using *Z-def* **by** *simp*
have *len-hp0*: $length\ hp0 = Suc\ T'$
using *assms* **by** *simp*
have *len-hp1*: $length\ hp1 = Suc\ T'$
using *assms* **by** *simp*
have *hp0*: $\forall i{<}length\ hp0.\ hp0\ !\ i \leq T'$
proof *standard+*
fix *i* :: *nat*
assume $i < length\ hp0$
then have $hp0\ !\ i = exc\ (zeros\ m)\ i <\#> 0$
using *map-nth assms*(*10*) **by** *blast*
then show $hp0\ !\ i \leq T'$
using *inputpos-less inputpos-def* **by** *simp*
qed
have *hp1*: $\forall i{<}length\ hp1.\ hp1\ !\ i \leq T'$
proof *standard+*
fix *i* :: *nat*
assume $i < length\ hp1$
then have $hp1\ !\ i = exc\ (zeros\ m)\ i <\#> 1$
using *map-nth assms*(*11*) **by** *blast*
then show $hp1\ !\ i \leq T'$
using *headpos-1-less* **by** *simp*
qed
have *psi*: $variables\ \psi \subseteq \{..<3*Z+H\}\ fsize\ \psi \leq (3*Z+H) * 2\ \hat{}\ (3*Z+H)\ length\ \psi \leq 2\ \hat{}\ (3*Z+H)$
using *psi* **by** *simp-all*
have *psi′*: $variables\ \psi' \subseteq \{..<2*Z+H\}\ fsize\ \psi' \leq (2*Z+H) * 2\ \hat{}\ (2*Z+H)\ length\ \psi' \leq 2\ \hat{}\ (2*Z+H)$
using *psi′* **by** *simp-all*

let $?sigma = [N + (t - 1) * Z..<N + (t - 1) * Z + Z]$ @
$(if\ previous\ hp1\ t \neq t\ then\ [N + previous\ hp1\ t * Z..<N + previous\ hp1\ t * Z + Z]\ else\ [])$ @

```
  [N + t * Z..<N + t * Z + Z] @
  [hp0 ! t * H..<hp0 ! t * H + H]

have hp0-nth: hp0 ! i = exc (zeros m) i <#> 0 if i ≤ T' for i
  using that assms map-nth len-hp0 by (metis (no-types, lifting) le-imp-less-Suc)
then have hp0-eq-inputpos: hp0 ! i = inputpos m i if i ≤ T' for i
  using inputpos-def that by simp

have hp1-nth: hp1 ! i = exc (zeros m) i <#> 1 if i ≤ T' for i
  using that assms map-nth len-hp1 by (metis (no-types, lifting) le-imp-less-Suc)

have previous-eq-prev: previous hp1 idx = prev m idx if idx ≤ T' for idx
proof (cases ∃i<idx. hp1 ! i = hp1 ! idx)
  case True
  then have 1: ∃i<idx. exc (zeros m) i <#> 1 = exc (zeros m) idx <#> 1
    using that hp1-nth by auto
  then have prev m idx = (GREATEST i. i < idx ∧ exc (zeros m) i <#> 1 = exc (zeros m) idx <#> 1)
    using prev-def by simp
  have previous hp1 idx = (GREATEST i. i < idx ∧ hp1 ! i = hp1 ! idx)
    using True previous-def by simp
  also have ... = (GREATEST i. i < idx ∧ exc (zeros m) i <#> 1 = exc (zeros m) idx <#> 1)
    (is Greatest ?P = Greatest ?Q)
  proof (rule Greatest-equality)
    have ∃i. ?Q i
      using 1 by simp
    moreover have 2: ∀y. ?Q y ⟶ y ≤ idx
      by simp
    ultimately have 3: ?Q (Greatest ?Q)
      using GreatestI-ex-nat[of ?Q] by blast
    then have 4: Greatest ?Q < idx
      by simp
    then have Greatest ?Q ≤ T'
      using that by simp
    then have hp1 ! (Greatest ?Q) = exc (zeros m) (Greatest ?Q) <#> 1
      using hp1-nth by simp
    moreover have hp1 ! idx = exc (zeros m) idx <#> 1
      using that hp1-nth by simp
    ultimately have hp1 ! (Greatest ?Q) = hp1 ! idx
      using 3 by simp
    then show ?P (Greatest ?Q)
      using 4 by simp
    show i ≤ Greatest ?Q if ?P i for i
    proof -
      have i < idx
        using that by simp
      then have hp1 ! i = exc (zeros m) i <#> 1
        using ‹idx ≤ T'› hp1-nth by simp
      moreover have hp1 ! idx = exc (zeros m) idx <#> 1
        using ‹idx ≤ T'› hp1-nth by simp
      ultimately have exc (zeros m) i <#> 1 = exc (zeros m) idx <#> 1
        using that by simp
      then have ?Q i
        using ‹i < idx› by simp
      then show ?thesis
        using Greatest-le-nat[of ?Q i] 2 by blast
    qed
  qed
  also have ... = prev m idx
    using prev-def 1 by simp
  finally show ?thesis .
next
  case False
  have ¬ (∃i<idx. exc (zeros m) i <#> 1 = exc (zeros m) idx <#> 1)
```

```
  proof (rule ccontr)
    assume ¬ (¬ (∃ i<idx. exc (zeros m) i <#> 1 = exc (zeros m) idx <#> 1))
    then obtain i where i < idx exc (zeros m) i <#> 1 = exc (zeros m) idx <#> 1
      by auto
    then have hp1 ! i = hp1 ! idx
      using hp1-nth that by simp
    then show False
      using False ‹i < idx› by simp
  qed
  then have prev m idx = idx
    using prev-def by auto
  moreover have previous hp1 idx = idx
    using False assms previous-def by auto
  ultimately show ?thesis
    by simp
qed

have ζ₀ i @ ζ₁ i @ ζ₂ i = [N + i ∗ Z..<N + (Suc i) ∗ Z] for i
  using zeta0-def zeta1-def zeta2-def upt-append-upt Z by simp
then have zeta012: ζ₀ i @ ζ₁ i @ ζ₂ i = [N + i ∗ Z..<N + i ∗ Z + Z] for i
  by (simp add: ab-semigroup-add-class.add-ac(1) add.commute[of Z i∗Z])
have gamma: γ (inputpos m i) = [inputpos m i ∗ H..<inputpos m i ∗ H + H] for i
  using gamma-def by (simp add: add.commute)

have rho: ϱ t = ?sigma if prev m t < t
proof −
  have previous hp1 t ≠ t
    using t that previous-eq-prev by simp
  then have ?sigma = [N + (t − 1) ∗ Z..<N + (t − 1) ∗ Z + Z] @
      [N + prev m t ∗ Z..<N + prev m t ∗ Z + Z] @
      [N + t ∗ Z..<N + t ∗ Z + Z] @
      [inputpos m t ∗ H..<inputpos m t ∗ H + H]
    using previous-eq-prev hp0-eq-inputpos t by simp
  also have ... = (ζ₀ (t − 1) @ ζ₁ (t − 1) @ ζ₂ (t − 1)) @
      (ζ₀ (prev m t) @ ζ₁ (prev m t) @ ζ₂ (prev m t)) @
      (ζ₀ t @ ζ₁ t @ ζ₂ t) @
      γ (inputpos m t)
    using zeta012 gamma by simp
  also have ... = ϱ t
    using rho-def by simp
  finally have ?sigma = ϱ t .
  then show ?thesis
    by simp
qed

have rho': ϱ' t = ?sigma if prev m t = t
proof −
  have previous hp1 t = t
    using t that previous-eq-prev by simp
  then have ?sigma = [N + (t − 1) ∗ Z..<N + (t − 1) ∗ Z + Z] @
      [N + t ∗ Z..<N + t ∗ Z + Z] @
      [inputpos m t ∗ H..<inputpos m t ∗ H + H]
    using previous-eq-prev hp0-eq-inputpos t by simp
  also have ... = (ζ₀ (t − 1) @ ζ₁ (t − 1) @ ζ₂ (t − 1)) @
      (ζ₀ t @ ζ₁ t @ ζ₂ t) @
      γ (inputpos m t)
    using zeta012 gamma by simp
  also have ... = ϱ' t
    using rho'-def by simp
  finally have ?sigma = ϱ' t .
  then show ?thesis
    by simp
qed
```

have χ t = *relabel ?sigma* (*if previous hp1 t = t then* ψ' *else* ψ)
proof (*cases prev m t < t*)
  case *True*
  then have χ t = *relabel* (ϱ *t*) ψ
    using *chi-def* **by** *simp*
  moreover have *previous hp1 t < t*
    using *t True previous-eq-prev* **by** *simp*
  ultimately show *?thesis*
    using *rho True* **by** *simp*
next
  case *False*
  then have $*$: *prev m t = t*
    by (*simp add*: *nat-less-le prev-le*)
  then have χ t = *relabel* (ϱ' *t*) ψ'
    using *chi-def* **by** *simp*
  moreover have *previous hp1 t = t*
    using *t* $*$ *previous-eq-prev* **by** *simp*
  ultimately show *?thesis*
    using *rho'* $*$ **by** *simp*
qed

then show *transforms* (*tm-chi j1 j2 j3*) *tps0 ttt tps'*
 using *transforms-tm-chi*[*OF jk G Z N len-hp0 hp0 len-hp1 hp1 t T T'-less psi psi' tps0 - assms*(*24*)] *assms*(*23*)
  by *simp*
qed

lemma *Z-sq-le*: $Z\hat{}2 \le 2\hat{}(16{*}Z)$
proof −
  have $Z\hat{}2 \le 2\hat{}(2{*}Z)$
    using *pow2-le-2pow2*[*of Z*] **by** *simp*
  also have ... $\le 2\hat{}(16{*}Z)$
    by *simp*
  finally show $Z\hat{}2 \le 2\hat{}(16{*}Z)$ **.**
qed

lemma *tm-PHI9* [*transforms-intros*]:
  fixes *tps0 tps'* :: *tape list* **and** *k* :: *nat* **and** *j1 j2 j3* :: *tapeidx*
  fixes *nss* :: *nat list list*
  assumes *jk*: *length tps0 = k 1 < j1 j1 < j2 j2 < j3 j3 + 17 < k*
  assumes *hp0 = map* (λ*t. exc* (*zeros m*) *t* <#> *0*) [*0..*<*Suc T'*]
  assumes *hp1 = map* (λ*t. exc* (*zeros m*) *t* <#> *1*) [*0..*<*Suc T'*]
  assumes *tps0*:
    tps0 ! *1* = *nlltape nss*
    tps0 ! *j1* = ($\lfloor hp0 \rfloor_{NL}$, *1*)
    tps0 ! *j2* = ($\lfloor hp1 \rfloor_{NL}$, *1*)
    tps0 ! *j3* = ($\lfloor N \rfloor_{N}$, *1*)
    tps0 ! (*j3* + *1*) = ($\lfloor H \rfloor_{N}$, *1*)
    tps0 ! (*j3* + *2*) = ($\lfloor Z \rfloor_{N}$, *1*)
    tps0 ! (*j3* + *3*) = ($\lfloor T' \rfloor_{N}$, *1*)
    tps0 ! (*j3* + *4*) = ($\lfloor$*formula-n psi*$\rfloor_{NLL}$, *1*)
    tps0 ! (*j3* + *5*) = ($\lfloor$*formula-n psi'*$\rfloor_{NLL}$, *1*)
    tps0 ! (*j3* + *6*) = ($\lfloor 1 \rfloor_{N}$, *1*)
    $\bigwedge$*i. 6 < i* $\Longrightarrow$ *i < 17* $\Longrightarrow$ *tps0* ! (*j3* + *i*) = ($\lfloor [] \rfloor$, *1*)
  assumes *tps'*: *tps' = tps0*
      [*1* := *nlltape* (*nss* @ *formula-n* Φ_9),
       j3 + *6* := ($\lfloor$*Suc T'*$\rfloor_{N}$, *1*),
       j3 + *3* := ($\lfloor 0 \rfloor_{N}$, *1*)]
  assumes *ttt* = *16114767* $*$ *2* ^ (*16* $*$ *Z*) $*$ *N* ^ *7*
  shows *transforms* (*tm-PHI9 j1 j2 j3*) *tps0 ttt tps'*
proof −
  define *tps* **where** *tps* = (λ*t. tps0*
      [*1* := *nlltape* (*nss* @ *concat* (*map* (λ*t. formula-n* (χ (*Suc t*))) [*0..*<*t*])),

```
        j3 + 6 := (⌊Suc t⌋N, 1),
        j3 + 3 := (⌊T' − t⌋N, 1)])
  have transforms (tm-PHI9 j1 j2 j3) (tps 0) ttt (tps T')
    unfolding tm-PHI9-def
  proof (tform)
    let ?ttt = 16114765 ∗ 2^(16∗Z) ∗ N ^ 6
    show transforms (tm-chi j1 j2 j3) (tps i) ?ttt (tps (Suc i)) if i < T'  for i
    proof (rule tm-chi ; (use assms tps-def that in simp ; fail)?)
      show tps (Suc i) = (tps i)
          [1 := nlltape
                ((nss @ concat (map (λt. formula-n (χ (Suc t))) [0..<i])) @
                  formula-n (χ (Suc i))),
          j3 + 6 := (⌊Suc (Suc i)⌋N, 1),
          j3 + 3 := (⌊T' − i − 1⌋N, 1)]
        using that tps-def by (simp add: list-update-swap)
    qed
    show ⋀i. i < T' ⟹ read (tps i) ! (j3 + 3) ≠ □
      using jk tps-def read-ncontents-eq-0 by simp
    show ¬ read (tps T') ! (j3 + 3) ≠ □
      using jk tps-def read-ncontents-eq-0 by simp
    show T' ∗ (16114765 ∗ 2 ^ (16 ∗ Z) ∗ N ^ 6 + 2) + 1 ≤ ttt
    proof −
      have T' ∗ (16114765 ∗ 2 ^ (16 ∗ Z) ∗ N ^ 6 + 2) + 1 ≤ T' ∗ (16114767 ∗ 2 ^ (16 ∗ Z) ∗ N ^ 6) + 1
        using Z-sq-le H-gr-2 N-eq by auto
      also have ... ≤ T' ∗ (16114767 ∗ 2 ^ (16 ∗ Z) ∗ N ^ 6) + (16114767 ∗ 2 ^ (16 ∗ Z) ∗ N ^ 6)
        using H-gr-2 N-eq by auto
      also have ... = Suc T' ∗ (16114767 ∗ 2 ^ (16 ∗ Z) ∗ N ^ 6)
        by simp
      also have ... ≤ N ∗ (16114767 ∗ 2 ^ (16 ∗ Z) ∗ N ^ 6)
        using T'-less Suc-leI mult-le-mono1 by presburger
      also have ... = 16114767 ∗ 2 ^ (16 ∗ Z) ∗ N ^ 7
        by algebra
      also have ... = ttt
        using assms(20) by simp
      finally show ?thesis .
    qed
  qed
  moreover have tps 0 = tps0
    using tps-def tps0 list-update-id[of tps0 Suc 0] list-update-id[of tps0 j3 + 6]
      list-update-id[of tps0 j3 + 3]
    by simp
  moreover have tps T' = tps'
  proof −
    have concat (map (λt. formula-n (χ (Suc t))) [0..<T']) =
        formula-n (concat (map (λt. χ (Suc t)) [0..<T']))
      using concat-formula-n by simp
    then show ?thesis
      using PHI9-def tps-def tps' list-update-id[of tps0 Suc 0] list-update-id[of tps0 j3 + 6]
        list-update-id[of tps0 j3 + 3]
      by simp
  qed
  ultimately show transforms (tm-PHI9 j1 j2 j3) tps0 ttt tps'
    by simp
qed

end

end
```

Chapter 8

Turing machines for reducing $\mathcal{NP}$ languages to SAT

theory *Reduction-TM*
  imports *Sat-TM-CNF Oblivious-2-Tape*
begin

At long last we are going to create a polynomial-time Turing machine that, for a fixed language $L \in \mathcal{NP}$, computes for every string x a CNF formula Φ such that $x \in L$ iff. Φ is satisfiable. This concludes the proof of the Cook-Levin theorem.

The CNF formula Φ is a conjunction of formulas $\Phi_0, \ldots, \Phi_9$, and the previous chapter has provided us with Turing machines *tm-PHI0*, *tm-PHI1*, etc. that are supposed to generate these formulas. But only for Φ_9 has this been proven yet. So our first task is to transfer the Turing machines *tm-PHI0*, ..., *tm-PHI8* into the locale *reduction-sat-x* and show that they really do generate the CNF formulas $\Phi_0, \ldots, \Phi_8$.

The TMs require certain values on their tapes prior to starting. Therefore we build a Turing machine that computes these values. Then, in a final effort, we combine all these TMs to create this article's biggest Turing machine.

8.1 Turing machines for parts of Φ revisited

In this section we restate the semantic lemmas *transforms-tm-PHI0* etc. of the Turing machines *tm-PHI0* etc. in the context of the locale *reduction-sat-x*. This means that the lemmas now have terms like *formula-n* Φ_0 in them instead of more complicated expressions. It also means that we more clearly see which values the tapes need to contain initially because they are now expressed in terms of values in the locale, such as n, $p(n)$, or m'.

context *reduction-sat-x*
begin

lemma *tm-PHI0* [*transforms-intros*]:
  fixes $tps\ tps' :: tape\ list$ **and** $j :: tapeidx$ **and** $ttt\ k :: nat$
  assumes $length\ tps = k$ **and** $1 < j$ **and** $j + 8 < k$
  assumes
    $tps\ !\ 1 = (\lfloor[]\rfloor,\ 1)$
    $tps\ !\ j = (\lfloor m'\rfloor_N,\ 1)$
    $tps\ !\ (j + 1) = (\lfloor H\rfloor_N,\ 1)$
    $tps\ !\ (j + 2) = (\lfloor[]\rfloor,\ 1)$
    $tps\ !\ (j + 3) = (\lfloor[]\rfloor,\ 1)$
    $tps\ !\ (j + 4) = (\lfloor[]\rfloor,\ 1)$
    $tps\ !\ (j + 5) = (\lfloor[]\rfloor,\ 1)$
    $tps\ !\ (j + 6) = (\lfloor[]\rfloor,\ 1)$
    $tps\ !\ (j + 7) = (\lfloor[]\rfloor,\ 1)$
    $tps\ !\ (j + 8) = (\lfloor[]\rfloor,\ 1)$
  assumes $tps' = tps$

$[j := (\lfloor Suc\ (Suc\ m')\rfloor_N,\ 1),$
$\ j + 2 := (\lfloor 0\rfloor_N,\ 1),$
$\ j + 6 := (\lfloor nll\text{-}Psi\ (Suc\ (Suc\ m') * H)\ H\ 0\rfloor_{NLL},\ 1),$
$\ 1 := nlltape\ (formula\text{-}n\ \Phi_0)]$
assumes $ttt = 5627 * H \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2$
shows *transforms* (*tm-PHI0 j*) *tps ttt tps'*
proof −
have $nll\text{-}Psi\ (m' * H)\ H\ 1 = formula\text{-}n\ (\Psi\ (\zeta_0\ 0)\ 1)$
using *nll-Psi zeta0-def m'* **by** *simp*
moreover have $nll\text{-}Psi\ (H + m' * H)\ H\ 1 = formula\text{-}n\ (\Psi\ (\zeta_1\ 0)\ 1)$
using *nll-Psi zeta1-def m'*
by (*smt* (*verit*) *ab-semigroup-add-class.add-ac*(*1*) *add.commute add-cancel-left-right mult-2 mult-zero-left*)
moreover have $nll\text{-}Psi\ (Suc\ (Suc\ m') * H)\ H\ 0 = formula\text{-}n\ (\Psi\ (\zeta_2\ 0)\ 0)$
proof −
have $Suc\ (Suc\ m') * H = N + 2 * H$
using *m'* **by** *simp*
moreover have $Suc\ (Suc\ m') * H + H = N + (Suc\ 0) * Z$
using *m' Z-def* **by** *simp*
ultimately have $\zeta_2\ 0 = [Suc\ (Suc\ m') * H..{<}Suc\ (Suc\ m') * H + H]$
using *zeta2-def* **by** (*metis Nat.add-0-right mult-zero-left*)
then show *?thesis*
using *nll-Psi* **by** *simp*
qed
ultimately have $nll\text{-}Psi\ (m' * H)\ H\ 1\ @\ nll\text{-}Psi\ (H + m' * H)\ H\ 1\ @\ nll\text{-}Psi\ (Suc\ (Suc\ m') * H)\ H\ 0 =$
$formula\text{-}n\ \Phi_0$
using *formula-n-def PHI0-def* **by** *simp*
then show *?thesis*
using *transforms-tm-PHI0I*[*OF assms*(*1*−*3*) *H-ge-3 assms*(*4*−*13*)] *assms*(*14*,*15*) **by** *simp*
qed

lemma *tm-PHI1* [*transforms-intros*]:
fixes *tps tps'* :: *tape list* **and** *j* :: *tapeidx* **and** *ttt k* :: *nat* **and** *nss* :: *nat list list*
assumes $length\ tps = k$ **and** $1 < j$ **and** $j + 7 < k$
assumes
$tps\ !\ 1 = nlltape\ nss$
$tps\ !\ j = (\lfloor 0\rfloor_N,\ 1)$
$tps\ !\ (j + 1) = (\lfloor H\rfloor_N,\ 1)$
$tps\ !\ (j + 2) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 3) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 4) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 5) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 6) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 7) = (\lfloor[]\rfloor,\ 1)$
assumes $tps' = tps$
$[j + 2 := (\lfloor 1\rfloor_N,\ 1),$
$\ j + 6 := (\lfloor nll\text{-}Psi\ 0\ H\ 1\rfloor_{NLL},\ 1),$
$\ 1 := nlltape\ (nss\ @\ formula\text{-}n\ \Phi_1)]$
assumes $ttt = 1875 * H \hat{}\ 4$
shows *transforms* (*tm-PHI1 j*) *tps ttt tps'*
proof −
have $nll\text{-}Psi\ 0\ H\ 1 = formula\text{-}n\ (\Psi\ ([0..{<}H])\ 1)$
using *nll-Psi* **by** *simp*
then have $nll\text{-}Psi\ 0\ H\ 1 = formula\text{-}n\ (\Psi\ (\gamma\ 0)\ 1)$
using *gamma-def* **by** *simp*
then have $nll\text{-}Psi\ 0\ H\ 1 = formula\text{-}n\ \Phi_1$
using *PHI1-def* **by** *simp*
then show *?thesis*
using *transforms-tm-PHI1I*[*OF assms*(*1*−*3*) *H-ge-3 assms*(*4*−*12*)] *assms*(*13*,*14*) **by** *simp*
qed

lemma *tm-PHI2* [*transforms-intros*]:
fixes *tps tps'* :: *tape list* **and** *j* :: *tapeidx* **and** *ttt k* :: *nat* **and** *nss* :: *nat list list*
assumes $length\ tps = k$ **and** $1 < j$ **and** $j + 8 < k$

assumes $idx = n$
assumes
$tps ! 1 = nlltape\ nss$
$tps ! j = (\lfloor idx \rfloor_N, 1)$
$tps ! (j + 1) = (\lfloor H \rfloor_N, 1)$
$tps ! (j + 2) = (\lfloor [] \rfloor, 1)$
$tps ! (j + 3) = (\lfloor [] \rfloor, 1)$
$tps ! (j + 4) = (\lfloor [] \rfloor, 1)$
$tps ! (j + 5) = (\lfloor [] \rfloor, 1)$
$tps ! (j + 6) = (\lfloor [] \rfloor, 1)$
$tps ! (j + 7) = (\lfloor [] \rfloor, 1)$
$tps ! (j + 8) = (\lfloor [] \rfloor, 1)$
assumes $ttt = 3764 * H \hat{}\ 4 * (3 + nlength\ (3 * H + 2 * idx * H))^2$
assumes $tps' = tps$
$[j := (\lfloor 2 * idx + 2 \rfloor_N, 1),$
$j + 2 := (\lfloor 3 \rfloor_N, 1),$
$j + 6 := (\lfloor nll\text{-}Psi\ (Suc\ (Suc\ (2 * idx)) * H)\ H\ 3 \rfloor_{NLL}, 1),$
$1 := nlltape\ (nss\ @\ formula\text{-}n\ \Phi_2)]$
shows *transforms* (*tm-PHI2* *j*) *tps ttt tps'*
proof −
have $nll\text{-}Psi\ (H + 2 * idx * H)\ H\ 3\ @\ nll\text{-}Psi\ (2 * H + 2 * idx * H)\ H\ 3 = formula\text{-}n\ \Phi_2$
proof −
have $\gamma\ (2 * n + 1) = [H + 2 * idx * H..<H + 2 * idx * H + H]$
using *assms*(*4*) *gamma-def* **by** *simp*
moreover have $\gamma\ (2 * n + 2) = [2 * H + 2 * idx * H..<2 * H + 2 * idx * H + H]$
using *assms*(*4*) *gamma-def* **by** *simp*
ultimately show $nll\text{-}Psi\ (H + 2 * idx * H)\ H\ 3\ @\ nll\text{-}Psi\ (2 * H + 2 * idx * H)\ H\ 3 = formula\text{-}n\ \Phi_2$
using *nll-Psi PHI2-def formula-n-def* **by** *simp*
qed
then show *?thesis*
using *transforms-tm-PHI2I*[*OF assms*(*1*−*3*) *H-ge-3 assms*(*5*−*14*)] *assms*(*15*,*16*) **by** *simp*
qed

lemma *PHI3-correct*: $concat\ (map\ (\lambda i.\ nll\text{-}Psi\ (H * (1 + 2 * i))\ H\ 2)\ [0..<n]) = formula\text{-}n\ \Phi_3$
proof −
have $nll\text{-}Psi\ (H * (1 + 2 * i))\ H\ 2 = formula\text{-}n\ (\Psi\ (\gamma\ (2{*}i{+}1))\ 2)$ **for** *i*
proof −
have $\gamma\ (2 * i + 1) = [H * (1 + 2 * i)..<H * (1 + 2 * i) + H]$
using *gamma-def* **by** (*simp add*: *mult.commute*)
then show *?thesis*
using *nll-Psi* **by** *simp*
qed
then have $concat\ (map\ (\lambda i.\ nll\text{-}Psi\ (H * (1 + 2 * i))\ H\ 2)\ [0..<n]) =$
$concat\ (map\ (\lambda i.\ formula\text{-}n\ (\Psi\ (\gamma\ (2{*}i{+}1))\ 2))\ [0..<n])$
by *simp*
also have $... = formula\text{-}n\ (concat\ (map\ (\lambda i.\ \Psi\ (\gamma\ (2{*}i{+}1))\ 2)\ [0..<n]))$
using *concat-formula-n* **by** *simp*
also have $... = formula\text{-}n\ \Phi_3$
using *PHI3-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *tm-PHI3*:
fixes *tps tps'* :: *tape list* **and** *j* :: *tapeidx* **and** *ttt k* :: *nat* **and** *nss* :: *nat list list*
assumes $length\ tps = k$ **and** $1 < j$ **and** $j + 8 < k$
assumes
$tps ! 1 = nlltape\ nss$
$tps ! j = (\lfloor 1 \rfloor_N, 1)$
$tps ! (j + 1) = (\lfloor H \rfloor_N, 1)$
$tps ! (j + 2) = (\lfloor 2 \rfloor_N, 1)$
$tps ! (j + 3) = (\lfloor [] \rfloor, 1)$
$tps ! (j + 4) = (\lfloor [] \rfloor, 1)$
$tps ! (j + 5) = (\lfloor [] \rfloor, 1)$

$tps\ !\ (j + 6) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 7) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 8) = (\lfloor 1 + 2 * n\rfloor_N,\ 1)$
assumes $ttt = Suc\ n * (9 + 1897 * (H \hat{}\ 4 * (nlength\ (1 + 2 * n))^2))$
assumes $tps' = tps$
$[j := (\lfloor 1 + 2 * n\rfloor_N,\ 1),$
$1 := nlltape\ (nss\ @\ formula\text{-}n\ \Phi_3),$
$j + 3 := (\lfloor 1\rfloor_N,\ 1)]$
shows *transforms (tm-PHI345 2 j) tps ttt tps'*
using *transforms-tm-PHI345I[OF assms(1,2,3) H-ge-3, of 2 2 nss 1 n] H-gr-2 assms PHI3-correct*
by *fastforce*

lemma *PHI4-correct*:
assumes $idx = 2 * n + 2 + 1$ **and** $kappa = 2$ **and** $step = 2$ **and** $numiter = p\ n$
shows $concat\ (map\ (\lambda i.\ nll\text{-}Psi\ (H * (idx + step * i))\ H\ kappa)\ [0..<numiter]) = formula\text{-}n\ \Phi_4$
proof −
have $nll\text{-}Psi\ (H * (idx + step * i))\ H\ kappa = formula\text{-}n\ (\Psi\ (\gamma\ (2 * n + 2 + 2 * i + 1))\ 2)$ **for** i
proof −
have $\gamma\ (2 * n + 2 + 2 * i + 1) = [H * (idx + step * i)..<H * (idx + step * i) + H]$
using *assms gamma-def* **by** (*simp add*: *add.commute mult.commute*)
then show *?thesis*
using *nll-Psi assms* **by** *simp*
qed
then have $concat\ (map\ (\lambda i.\ nll\text{-}Psi\ (H * (idx + step * i))\ H\ kappa)\ [0..<numiter]) =$
$concat\ (map\ (\lambda i.\ formula\text{-}n\ (\Psi\ (\gamma\ (2 * n + 2 + 2 * i + 1))\ 2))\ [0..<numiter])$
by *simp*
also have $... = formula\text{-}n\ (concat\ (map\ (\lambda i.\ \Psi\ (\gamma\ (2 * n + 2 + 2 * i + 1))\ 2)\ [0..<p\ n]))$
using *assms concat-formula-n* **by** *simp*
also have $... = formula\text{-}n\ \Phi_4$
using *PHI4-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *tm-PHI4*:
fixes *tps tps'* :: *tape list* **and** *j* :: *tapeidx* **and** *ttt step k* :: *nat* **and** *nss* :: *nat list list*
assumes $length\ tps = k$ **and** $1 < j$ **and** $j + 8 < k$ **assumes**
$tps\ !\ 1 = nlltape\ nss$
$tps\ !\ j = (\lfloor 2 * n + 2 + 1\rfloor_N,\ 1)$
$tps\ !\ (j + 1) = (\lfloor H\rfloor_N,\ 1)$
$tps\ !\ (j + 2) = (\lfloor 2\rfloor_N,\ 1)$
$tps\ !\ (j + 3) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 4) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 5) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 6) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 7) = (\lfloor[]\rfloor,\ 1)$
$tps\ !\ (j + 8) = (\lfloor 2 * n + 2 + 1 + 2 * p\ n\rfloor_N,\ 1)$
assumes $ttt = Suc\ (p\ n) * (9 + 1897 * (H \hat{}\ 4 * (nlength\ (2 * n + 2 + 1 + 2 * p\ n))^2))$
assumes $tps' = tps$
$[j := (\lfloor 2 * n + 2 + 1 + 2 * p\ n\rfloor_N,\ 1),$
$1 := nlltape\ (nss\ @\ formula\text{-}n\ \Phi_4),$
$j + 3 := (\lfloor 1\rfloor_N,\ 1)]$
shows *transforms (tm-PHI345 2 j) tps ttt tps'*
using *transforms-tm-PHI345I[OF assms(1,2,3) H-ge-3, of 2 2 nss 2 * n + 2 + 1 p n] H-gr-2 assms PHI4-correct*
by *fastforce*

lemma *PHI5-correct*:
assumes $idx = 2 * n + 2 * p\ n + 3$ **and** $kappa = 0$ **and** $step = 1$ **and** $numiter = T'$
shows $concat\ (map\ (\lambda i.\ nll\text{-}Psi\ (H * (idx + step * i))\ H\ kappa)\ [0..<numiter]) = formula\text{-}n\ \Phi_5$
proof −
have $nll\text{-}Psi\ (H * (idx + step * i))\ H\ kappa = formula\text{-}n\ (\Psi\ (\gamma\ (2 * n + 2 * p\ n + 3 + i))\ 0)$ **for** i
proof −
have $\gamma\ (2 * n + 2 * p\ n + 3 + i) = [H * (idx + step * i)..<H * (idx + step * i) + H]$

using *assms gamma-def* **by** (*simp add*: *add.commute mult.commute*)
then show *?thesis*
using *nll-Psi assms* **by** *simp*
qed
then have *concat* (*map* (λi. *nll-Psi* ($H * (idx + step * i)$) *H kappa*) [*0..<numiter*]) =
concat (*map* (λi. *formula-n* (Ψ (γ ($2 * n + 2 * p\ n + 3 + i$)) *0*)) [*0..<numiter*])
by *simp*
also have ... = *formula-n* (*concat* (*map* (λi. Ψ (γ ($2 * n + 2 * p\ n + 3 + i$)) *0*) [*0..<T′*]))
using *assms concat-formula-n* **by** *simp*
also have ... = *formula-n* Φ_5
using *PHI5-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *tm-PHI5*:
fixes *tps tps′* :: *tape list* **and** *j* :: *tapeidx* **and** *ttt k* :: *nat* **and** *nss* :: *nat list list*
assumes *length tps* = *k* **and** $1 < j$ **and** $j + 8 < k$
assumes
tps ! *1* = *nlltape nss*
tps ! *j* = ($\lfloor 2 * n + 2 * p\ n + 3 \rfloor_N$, *1*)
tps ! ($j + 1$) = ($\lfloor H \rfloor_N$, *1*)
tps ! ($j + 2$) = ($\lfloor 0 \rfloor_N$, *1*)
tps ! ($j + 3$) = ($\lfloor [] \rfloor$, *1*)
tps ! ($j + 4$) = ($\lfloor [] \rfloor$, *1*)
tps ! ($j + 5$) = ($\lfloor [] \rfloor$, *1*)
tps ! ($j + 6$) = ($\lfloor [] \rfloor$, *1*)
tps ! ($j + 7$) = ($\lfloor [] \rfloor$, *1*)
tps ! ($j + 8$) = ($\lfloor 2 * n + 2 * p\ n + 3 + T' \rfloor_N$, *1*)
assumes *ttt* = *Suc T′* $*$ ($9 + 1891 * (H$ ^ $4 * (nlength\ (2 * n + 2 * p\ n + 3 + T'))^2)$)
assumes *tps′* = *tps*
[*j* := ($\lfloor 2 * n + 2 * p\ n + 3 + T' \rfloor_N$, *1*),
1 := *nlltape* (*nss* @ *formula-n* Φ_5),
$j + 3$:= ($\lfloor 1 \rfloor_N$, *1*)]
shows *transforms* (*tm-PHI345 1 j*) *tps ttt tps′*
using *transforms-tm-PHI345I*[*OF assms*(*1,2,3*) *H-ge-3*, *of 0 1*, *OF - - assms*(*4−12*)] *H-gr-2 assms*(*13−*)
PHI5-correct
by *fastforce*

lemma *PHI6-correct*:
concat (*map* (λi. *nll-Psi* ($H * (2 + 2 * i)$) *H* (*xs* ! *i*)) [*0..<length xs*]) = *formula-n* Φ_6
proof −
have *nll-Psi* ($H * (2 + 2 * i)$) *H* (*xs* ! *i*) = *formula-n* (Ψ (γ ($2 * i + 2$)) (*if x* ! *i then 3 else 2*))
if $i <$ *length xs* **for** *i*
proof −
have γ ($2 * i + 2$) = [$H * (2 + 2 * i)$..<$H * (2 + 2 * i) + H$]
using *gamma-def* **by** (*simp add*: *mult.commute*)
then have *nll-Psi* ($H * (2 + 2 * i)$) *H* (*xs* ! *i*) = *formula-n* (Ψ (γ ($2 * i + 2$)) (*xs* ! *i*))
using *nll-Psi* **by** *simp*
moreover have *xs* ! *i* = (*if x* ! *i then 3 else 2*)
using *that* **by** *simp*
ultimately show *?thesis*
by *simp*
qed
then have *map* (λi. *nll-Psi* ($H * (2 + 2 * i)$) *H* (*xs* ! *i*)) [*0..<length xs*] =
map (λi. *formula-n* (Ψ (γ ($2 * i + 2$)) (*if x* ! *i then 3 else 2*))) [*0..<length xs*]
by *simp*
then have *concat* (*map* (λi. *nll-Psi* ($H * (2 + 2 * i)$) *H* (*xs* ! *i*)) [*0..<length xs*]) =
concat (*map* (λi. *formula-n* (Ψ (γ ($2 * i + 2$)) (*if x* ! *i then 3 else 2*))) [*0..<length xs*])
by *metis*
also have ... = *formula-n* (*concat* (*map* (λi. Ψ (γ ($2 * i + 2$)) (*if x* ! *i then 3 else 2*)) [*0..<length xs*]))
using *concat-formula-n* **by** *simp*
also have ... = *formula-n* (*concat* (*map* (λi. Ψ (γ ($2 * i + 2$)) (*if x* ! *i then 3 else 2*)) [*0..<n*]))
by *simp*

also have ... = *formula-n* Φ_6
using *PHI6-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *tm-PHI6* [*transforms-intros*]:
fixes *tps tps'* :: *tape list* **and** *j* :: *tapeidx* **and** *ttt k* :: *nat* **and** *nss* :: *nat list list*
assumes *length tps* = *k* **and** *1* < *j* **and** *j* + *7* < *k*
assumes
$tps\ !\ 1 = nlltape\ nss$
$tps\ !\ 0 = (\lfloor xs \rfloor,\ 1)$
$tps\ !\ j = (\lfloor 2 \rfloor_N,\ 1)$
$tps\ !\ (j + 1) = (\lfloor H \rfloor_N,\ 1)$
$tps\ !\ (j + 2) = (\lfloor 0 \rfloor_N,\ 1)$
$tps\ !\ (j + 3) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 4) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 5) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 6) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 7) = (\lfloor [] \rfloor,\ 1)$
assumes $tps' = tps$
$[0 := (\lfloor xs \rfloor,\ Suc\ n),$
$j := (\lfloor 2 + 2 * n \rfloor_N,\ 1),$
$1 := nlltape\ (nss\ @\ formula\text{-}n\ \Phi_6)]$
assumes $ttt = 133650 * H \hat{}\ 6 * n \hat{}\ 3 + 1$
shows *transforms* (*tm-PHI6 j*) *tps ttt tps'*
using *transforms-tm-PHI6I*[*OF assms*(*1*,*2*,*3*) *H-ge-3 bs-xs assms*(*4*−*13*) -] *assms*(*14*,*15*) *PHI6-correct*
by *simp*

lemma *PHI7-correct*:
assumes *idx* = *2* ∗ *n* + *4* **and** *numiter* = *p n*
shows *concat* (*map* (λ*i*. *nll-Upsilon* (*idx* + *2* ∗ *i*) *H*) [*0*..<*numiter*]) = *formula-n* Φ_7
proof −
have *nll-Upsilon* (*idx* + *2* ∗ *i*) *H* = *formula-n* (Υ (γ (*2*∗*n* + *4* + *2* ∗ *i*))) **for** *i*
proof −
have *nll-Upsilon* (*idx* + *2* ∗ *i*) *H* = *formula-n* (Υ [(*idx* + *2* ∗ *i*)∗*H*..<(*idx* + *2* ∗ *i*)∗*H*+*H*])
using *nll-Upsilon*[*OF H-ge-3*] **by** *simp*
also have ... = *formula-n* (Υ (γ (*2* ∗ *n* + *4* + *2* ∗ *i*)))
using *gamma-def assms*(*1*) **by** (*simp add*: *add.commute*)
finally show *?thesis* **.**
qed
then have *concat* (*map* (λ*i*. *nll-Upsilon* (*idx* + *2* ∗ *i*) *H*) [*0*..<*numiter*]) =
concat (*map* (λ*i*. *formula-n* (Υ (γ (*2*∗*n* + *4* + *2* ∗ *i*)))) [*0*..<*numiter*])
by *simp*
also have ... = *formula-n* (*concat* (*map* (λ*i*. Υ (γ (*2*∗*n* + *4* + *2* ∗ *i*))) [*0*..<*numiter*]))
using *concat-formula-n* **by** *simp*
also have ... = *formula-n* (*concat* (*map* (λ*i*. Υ (γ (*2*∗*n* + *4* + *2* ∗ *i*))) [*0*..<*p n*]))
using *assms*(*2*) **by** *simp*
also have ... = *formula-n* Φ_7
using *PHI7-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *tm-PHI7* [*transforms-intros*]:
fixes *tps tps'* :: *tape list* **and** *j* :: *tapeidx* **and** *ttt numiter k idx* :: *nat* **and** *nss* :: *nat list list*
assumes *length tps* = *k* **and** *1* < *j* **and** *j* + *6* < *k*
assumes
$tps\ !\ 1 = nlltape\ nss$
$tps\ !\ j = (\lfloor 2 * n + 4 \rfloor_N,\ 1)$
$tps\ !\ (j + 1) = (\lfloor H \rfloor_N,\ 1)$
$tps\ !\ (j + 2) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 3) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 4) = (\lfloor [] \rfloor,\ 1)$
$tps\ !\ (j + 5) = (\lfloor [] \rfloor,\ 1)$

$tps ! (j + 6) = (\lfloor p\ n \rfloor_N,\ 1)$
assumes $ttt = p\ n * 257 * H * (nlength\ (2 * n + 4 + 2 * p\ n) + nlength\ H)^2 + 1$
assumes $tps' = tps$
$[j := (\lfloor 2 * n + 4 + 2 * p\ n \rfloor_N,\ 1),$
$j + 6 := (\lfloor 0 \rfloor_N,\ 1),$
$1 := nlltape\ (nss\ @\ formula\text{-}n\ \Phi_7)]$
shows *transforms* (*tm-PHI7 j*) *tps ttt tps′*
using *transforms-tm-PHI7I*[*OF assms*(*1*,*2*,*3*) *H-ge-3 assms*(*4−12*)] *assms*(*13*) *PHI7-correct*
by *simp*

lemma *tm-PHI8* [*transforms-intros*]:
fixes *tps tps′* :: *tape list* **and** *j* :: *tapeidx* **and** *ttt k idx* :: *nat* **and** *nss* :: *nat list list*
assumes $length\ tps = k$ **and** $1 < j$ **and** $j + 7 < k$
assumes $idx = 1 + 3 * T' + m'$
assumes
$tps ! 1 = nlltape\ nss$
$tps ! j = (\lfloor 1 + 3 * T' + m' \rfloor_N,\ 1)$
$tps ! (j + 1) = (\lfloor H \rfloor_N,\ 1)$
$tps ! (j + 2) = (\lfloor [] \rfloor,\ 1)$
$tps ! (j + 3) = (\lfloor [] \rfloor,\ 1)$
$tps ! (j + 4) = (\lfloor [] \rfloor,\ 1)$
$tps ! (j + 5) = (\lfloor [] \rfloor,\ 1)$
$tps ! (j + 6) = (\lfloor [] \rfloor,\ 1)$
$tps ! (j + 7) = (\lfloor [] \rfloor,\ 1)$
assumes $tps' = tps$
$[1 := nlltape\ (nss\ @\ formula\text{-}n\ \Phi_8),$
$j + 2 := (\lfloor 3 \rfloor_N,\ 1),$
$j + 6 := (\lfloor formula\text{-}n\ \Phi_8 \rfloor_{NLL},\ 1)]$
assumes $ttt = 18 + 1861 * H\ \hat{}\ 4 * (nlength\ (Suc\ (1 + 3 * T' + m')))^2$
shows *transforms* (*tm-PHI8 j*) *tps ttt tps′*
proof −
let $?idx = 1 + 3 * T' + m'$
have $m' * H + T' * 3 * H + H = ?idx * H$
using *m′ add-mult-distrib* **by** *simp*
then have $\zeta_1\ T' = [?idx * H..{<}?idx * H + H]$
using *zeta1-def Z-def m′* **by** (*metis ab-semigroup-add-class.add-ac*(*1*) *mult.assoc mult-2*)
then have $nll\text{-}Psi\ (?idx * H)\ H\ 3 = formula\text{-}n\ \Phi_8$
using *PHI8-def nll-Psi* **by** *simp*
then show *?thesis*
using *transforms-tm-PHI8I*[*OF assms*(*1−3*) *H-ge-3 assms*(*5−13*) - *assms*(*15*)] *assms*(*14*) **by** *simp*
qed

end

8.2 A Turing machine for initialization

As we have seen in the previous section, the Turing machines *tm-PHI0* etc. expect some tapes to contain certain values that depend on the verifier TM M. In this section we construct the TM *tm-PHI-init* that computes theses values.

The TM expects the string x on the input tape. Then it determines the length n of x and stores it on tape 11. Then it computes the value $p(n)$ and stores it on tape 15. Then it computes $m = 2n + 2p(n) + 2$ and stores it on tape 16. It then writes $\mathbf{0}^m$ to tape 9 and runs *tm-list-headpos*, which writes the sequences of head positions for the input and work/output tape of the verifier TM M to tapes 4 and 7, respectively. The length of these lists determines T', which is written to tape 17. From this and m the TM computes m' and writes it to tape 18. It then writes H, which is hard-coded, to tape 19 and finally $N = H \cdot m'$ to tape 20.

We assume that the TM starts in a configuration where the input tape head and the heads on tapes with index greater than 10 are positioned on cell number 1, whereas all other tapes are on cell number 0 as usual. The TM has no tape parameters, as all tapes are fixed to work with the final TM later.

As with other TMs before, we will define and analyze the TM on the theory level and then transfer the semantics to the locale *reduction-sat-x*.

definition *tm-PHI-init* :: *nat* ⇒ *machine* ⇒ (*nat* ⇒ *nat*) ⇒ *machine* **where**
tm-PHI-init G M p ≡
tm-right 9 ;;
tm-length-input 11 ;;
tm-polynomial p 11 ;;
tm-copyn 15 16 ;;
tm-add 11 16 ;;
tm-incr 16 ;;
tm-times2 16 ;;
tm-copyn 16 17 ;;
tm-write-replicate 2 17 9 ;;
tm-left 9 ;;
tm-list-headpos G M 2 ;;
tm-count 4 17 4 ;;
tm-decr 17 ;;
tm-copyn 16 18 ;;
tm-incr 18 ;;
tm-add 17 18 ;;
tm-setn 19 (*max G* (*length M*)) ;;
tm-mult 18 19 20

lemma *tm-PHI-init-tm*:
fixes *H k*
assumes *turing-machine 2 G M* **and** *k* > *20* **and** *H* ≥ *Suc* (*length M*) **and** *H* ≥ *G*
assumes *H* ≥ *5*
shows *turing-machine k H* (*tm-PHI-init G M p*)
unfolding *tm-PHI-init-def*
using *assms turing-machine-sequential-turing-machine tm-right-tm tm-length-input-tm tm-polynomial-tm tm-copyn-tm tm-add-tm tm-incr-tm tm-times2-tm tm-write-replicate-tm tm-left-tm tm-list-headpos-tm tm-count-tm tm-decr-tm tm-setn-tm tm-mult-tm*
by *simp*

locale *turing-machine-PHI-init* =
fixes *G* :: *nat* **and** *M* :: *machine* **and** *p* :: *nat* ⇒ *nat*
begin

definition *tm3* ≡ *tm-right 9*
definition *tm4* ≡ *tm3* ;; *tm-length-input 11*
definition *tm5* ≡ *tm4* ;; *tm-polynomial p 11*
definition *tm6* ≡ *tm5* ;; *tm-copyn 15 16*
definition *tm7* ≡ *tm6* ;; *tm-add 11 16*
definition *tm8* ≡ *tm7* ;; *tm-incr 16*
definition *tm9* ≡ *tm8* ;; *tm-times2 16*
definition *tm10* ≡ *tm9* ;; *tm-copyn 16 17*
definition *tm11* ≡ *tm10* ;; *tm-write-replicate 2 17 9*
definition *tm12* ≡ *tm11* ;; *tm-left 9*
definition *tm13* ≡ *tm12* ;; *tm-list-headpos G M 2*
definition *tm14* ≡ *tm13* ;; *tm-count 4 17 4*
definition *tm15* ≡ *tm14* ;; *tm-decr 17*
definition *tm16* ≡ *tm15* ;; *tm-copyn 16 18*
definition *tm17* ≡ *tm16* ;; *tm-incr 18*
definition *tm18* ≡ *tm17* ;; *tm-add 17 18*
definition *tm19* ≡ *tm18* ;; *tm-setn 19* (*max G* (*length M*))
definition *tm20* ≡ *tm19* ;; *tm-mult 18 19 20*

lemma *tm20-eq-tm-PHI-init*: *tm20* = *tm-PHI-init G M p*
unfolding *tm20-def tm19-def tm18-def tm17-def tm16-def tm15-def tm14-def tm13-def tm12-def tm11-def*
unfolding *tm10-def tm9-def tm8-def tm7-def tm6-def tm5-def tm4-def tm3-def tm-PHI-init-def*
by *simp*

context
fixes *k H thalt* :: *nat* **and** *tps0* :: *tape list* **and** *xs zs* :: *symbol list*
assumes *poly-p*: *polynomial p*

and *M-tm*: *turing-machine 2 G M*
and *k*: *k* = *length tps0 20* < *k*
and *H*: *H* = *max G* (*length M*)
and *xs*: *bit-symbols xs*
and *zs*: *zs* = *2* # *2* # *replicate* (*2* ∗ *length xs* + *2* ∗ *p* (*length xs*)) *2*
assumes *thalt*:
∀ *t*<*thalt*. *fst* (*execute M* (*start-config 2 zs*) *t*) < *length M*
fst (*execute M* (*start-config 2 zs*) *thalt*) = *length M*
assumes *tps0*:
tps0 ! *0* = (⌊*xs*⌋, *1*)
⋀*i*. *0* < *i* ⟹ *i* ≤ *10* ⟹ *tps0* ! *i* = (⌊[]⌋, *0*)
⋀*i*. *10* < *i* ⟹ *i* < *k* ⟹ *tps0* ! *i* = (⌊[]⌋, *1*)
begin

lemma *G*: *G* ≥ *4*
using *M-tm turing-machine-def* **by** *simp*

lemma *H*: *H* ≥ *length M H* ≥ *G*
using *H* **by** *simp-all*

definition *tps3* ≡ *tps0*
[*9* := (⌊[]⌋, *1*)]

lemma *tm3* [*transforms-intros*]: *transforms tm3 tps0 1 tps3*
unfolding *tm3-def* **by** (*tform tps*: *tps3-def tps0 k*)

abbreviation *n* ≡ *length xs*

definition *tps4* ≡ *tps0*
[*9* := (⌊[]⌋, *1*),
11 := (⌊*n*⌋$_N$, *1*)]

lemma *tm4* [*transforms-intros*]:
assumes *ttt* = *5* + *11* ∗ (*length xs*)^2
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform tps*: *tps3-def tps4-def tps0 k time*: *assms*)
show *proper-symbols xs*
using *xs* **by** *auto*
show *tps3* ! *11* = (⌊*0*⌋$_N$, *1*)
using *canrepr-0 tps3-def tps0 k* **by** *simp*
qed

definition *tps5* ≡ *tps0*
[*9* := (⌊[]⌋, *1*),
11 := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*)]

lemma *tm5* [*transforms-intros*]:
assumes *ttt* = *5* + *11* ∗ (*length xs*)^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength* (*length xs*))^2)
shows *transforms tm5 tps0 ttt tps5*
unfolding *tm5-def* **by** (*tform tps*: *canrepr-0 tps4-def tps5-def tps0 k poly-p time*: *assms*)

definition *tps6* ≡ *tps0*
[*9* := (⌊[]⌋, *1*),
11 := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*p n*⌋$_N$, *1*)]

lemma *tm6* [*transforms-intros*]:
assumes *ttt* = *19* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) + *3* ∗ *nlength* (*p n*)
shows *transforms tm6 tps0 ttt tps6*
unfolding *tm6-def*

proof (*tform tps*: *tps5-def tps6-def tps0 k*)
  show *tps5* ! *16* = ($\lfloor 0 \rfloor_N$, *1*)
    using *canrepr-0 k tps0 tps5-def* **by** *simp*
  show *ttt* = *5* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) +
      (*14* + *3* ∗ (*nlength* (*p n*) + *nlength 0*))
    using *assms* **by** *simp*
qed

definition *tps7* ≡ *tps0*
  [*9* := ($\lfloor[]\rfloor$, *1*),
   11 := ($\lfloor n \rfloor_N$, *1*),
   15 := ($\lfloor p\ n \rfloor_N$, *1*),
   16 := ($\lfloor n + p\ n \rfloor_N$, *1*)]

lemma *tm7* [*transforms-intros*]:
  assumes *ttt* = *29* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) +
    3 ∗ *nlength* (*p n*) + *3* ∗ *max* (*nlength n*) (*nlength* (*p n*))
  shows *transforms tm7 tps0 ttt tps7*
  unfolding *tm7-def* **by** (*tform tps*: *tps6-def tps7-def tps0 k assms*)

definition *tps8* ≡ *tps0*
  [*9* := ($\lfloor[]\rfloor$, *1*),
   11 := ($\lfloor n \rfloor_N$, *1*),
   15 := ($\lfloor p\ n \rfloor_N$, *1*),
   16 := ($\lfloor Suc\ (n + p\ n) \rfloor_N$, *1*)]

lemma *tm8* [*transforms-intros*]:
  assumes *ttt* = *34* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) +
    3 ∗ *nlength* (*p n*) + *3* ∗ *max* (*nlength n*) (*nlength* (*p n*)) + *2* ∗ *nlength* (*n* + *p n*)
  shows *transforms tm8 tps0 ttt tps8*
  unfolding *tm8-def* **by** (*tform tps*: *tps7-def tps8-def tps0 k assms*)

definition *tps9* ≡ *tps0*
  [*9* := ($\lfloor[]\rfloor$, *1*),
   11 := ($\lfloor n \rfloor_N$, *1*),
   15 := ($\lfloor p\ n \rfloor_N$, *1*),
   16 := ($\lfloor 2 * Suc\ (n + p\ n) \rfloor_N$, *1*)]

lemma *tm9* [*transforms-intros*]:
  assumes *ttt* = *39* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) +
    3 ∗ *nlength* (*p n*) + *3* ∗ *max* (*nlength n*) (*nlength* (*p n*)) + *2* ∗ *nlength* (*n* + *p n*) +
    2 ∗ *nlength* (*Suc* (*n* + *p n*))
  shows *transforms tm9 tps0 ttt tps9*
  unfolding *tm9-def* **by** (*tform tps*: *tps8-def tps9-def tps0 k assms*)

definition *tps10* ≡ *tps0*
  [*9* := ($\lfloor[]\rfloor$, *1*),
   11 := ($\lfloor n \rfloor_N$, *1*),
   15 := ($\lfloor p\ n \rfloor_N$, *1*),
   16 := ($\lfloor 2 * Suc\ (n + p\ n) \rfloor_N$, *1*),
   17 := ($\lfloor 2 * Suc\ (n + p\ n) \rfloor_N$, *1*)]

lemma *tm10* [*transforms-intros*]:
  assumes *ttt* = *53* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) +
    3 ∗ *nlength* (*p n*) + *3* ∗ *max* (*nlength n*) (*nlength* (*p n*)) + *2* ∗ *nlength* (*n* + *p n*) +
    2 ∗ *nlength* (*Suc* (*n* + *p n*)) + *3* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))
  shows *transforms tm10 tps0 ttt tps10*
  unfolding *tm10-def*
proof (*tform tps*: *tps9-def tps10-def tps0 k*)
  show *tps9* ! *17* = ($\lfloor 0 \rfloor_N$, *1*)
    using *tps9-def canrepr-0 tps0 k* **by** *simp*
  show *ttt* = *39* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) +
      3 ∗ *nlength* (*p n*) + *3* ∗ *max* (*nlength n*) (*nlength* (*p n*)) +

$2 * nlength\ (n + p\ n) + 2 * nlength\ (Suc\ (n + p\ n)) +$
$(14 + 3 * (nlength\ (Suc\ (Suc\ (2 * n + 2 * p\ n))) + nlength\ 0))$
using *assms* **by** *simp*
qed

definition $tps11 \equiv tps0$
$[9 := (\lfloor zs \rfloor, 1),$
$11 := (\lfloor n \rfloor_N, 1),$
$15 := (\lfloor p\ n \rfloor_N, 1),$
$16 := (\lfloor 2 * Suc\ (n + p\ n) \rfloor_N, 1),$
$17 := (\lfloor 0 \rfloor_N, 1)]$

lemma *tm11* [*transforms-intros*]:
assumes $ttt = 57 + 11 * n^2 + (d\text{-}polynomial\ p + d\text{-}polynomial\ p * (nlength\ n)^2) +$
$3 * nlength\ (p\ n) + 3 * max\ (nlength\ n)\ (nlength\ (p\ n)) + 2 * nlength\ (n + p\ n) +$
$2 * nlength\ (Suc\ (n + p\ n)) + 3 * nlength\ (Suc\ (Suc\ (2 * n + 2 * p\ n))) +$
$Suc\ (Suc\ (2 * n + 2 * p\ n)) * (12 + 2 * nlength\ (Suc\ (Suc\ (2 * n + 2 * p\ n))))$
shows *transforms tm11 tps0 ttt tps11*
unfolding *tm11-def*
proof (*tform tps*: *tps10-def tps11-def tps0 k time*: *assms*)
show $tps11 = tps10$
$[17 := (\lfloor 0 \rfloor_N, 1),$
$9 := (\lfloor replicate\ (Suc\ (Suc\ (2 * n + 2 * p\ n)))\ 2 \rfloor, 1)]$
unfolding *tps11-def tps10-def* **using** *zs* **by** (*simp add*: *list-update-swap*[*of - 9*])
qed

definition $tps12 \equiv tps0$
$[9 := (\lfloor zs \rfloor, 0),$
$11 := (\lfloor n \rfloor_N, 1),$
$15 := (\lfloor p\ n \rfloor_N, 1),$
$16 := (\lfloor 2 * Suc\ (n + p\ n) \rfloor_N, 1),$
$17 := (\lfloor 0 \rfloor_N, 1)]$

lemma *tm12* [*transforms-intros*]:
assumes $ttt = 82 + 11 * n^2 + (d\text{-}polynomial\ p + d\text{-}polynomial\ p * (nlength\ n)^2) +$
$3 * nlength\ (p\ n) + 3 * max\ (nlength\ n)\ (nlength\ (p\ n)) + 2 * nlength\ (n + p\ n) +$
$2 * nlength\ (Suc\ (n + p\ n)) + 7 * nlength\ (Suc\ (Suc\ (2 * n + 2 * p\ n))) +$
$(2 * n + 2 * p\ n) * (12 + 2 * nlength\ (Suc\ (Suc\ (2 * n + 2 * p\ n))))$
shows *transforms tm12 tps0 ttt tps12*
unfolding *tm12-def*
proof (*tform tps*: *tps11-def tps12-def tps0 k time*: *assms*)
have $tps11\ !\ 9\ |-|\ 1 = (\lfloor zs \rfloor, 0)$
using *tps11-def k* **by** *simp*
then show $tps12 = tps11[9 := tps11\ !\ 9\ |-|\ 1]$
unfolding *tps12-def tps11-def* **by** (*simp add*: *list-update-swap*[*of - 9*])
qed

abbreviation *exc* :: $nat \Rightarrow config$ **where**
$exc\ t \equiv execute\ M\ (start\text{-}config\ 2\ zs)\ t$

definition $tps13 \equiv tps0$
$[9 := exc\ thalt <!> 0,$
$11 := (\lfloor n \rfloor_N, 1),$
$15 := (\lfloor p\ n \rfloor_N, 1),$
$16 := (\lfloor 2 * Suc\ (n + p\ n) \rfloor_N, 1),$
$17 := (\lfloor 0 \rfloor_N, 1),$
$3 := (\lfloor exc\ thalt <\#> 0 \rfloor_N, 1),$
$4 := (\lfloor map\ (\lambda t.\ exc\ t <\#> 0)\ [0..<Suc\ thalt] \rfloor_{NL}, 1),$
$6 := (\lfloor exc\ thalt <\#> 1 \rfloor_N, 1),$
$7 := (\lfloor map\ (\lambda t.\ exc\ t <\#> 1)\ [0..<Suc\ thalt] \rfloor_{NL}, 1),$
$10 := exc\ thalt <!> 1]$

lemma *tm13* [*transforms-intros*]:

```
  assumes ttt = 82 + 11 ∗ n^2 + (d-polynomial p + d-polynomial p ∗ (nlength n)^2) +
    3 ∗ nlength (p n) + 3 ∗ max (nlength n) (nlength (p n)) + 2 ∗ nlength (n + p n) +
    2 ∗ nlength (Suc (n + p n)) + 7 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n))) +
    (2 ∗ n + 2 ∗ p n) ∗ (12 + 2 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n)))) +
    (27 + 27 ∗ thalt) ∗ (9 + 2 ∗ nlength thalt)
  shows transforms tm13 tps0 ttt tps13
  unfolding tm13-def
proof (tform)
  show turing-machine 2 G M
    using M-tm .
  show 2 + 9 ≤ length tps12
    using tps12-def k by simp
  show ∀ t<thalt. fst (execute M (start-config 2 zs) t) < length M
      fst (execute M (start-config 2 zs) thalt) = length M
    using thalt .
  show symbols-lt G zs
  proof −
    have zs = replicate (2 ∗ n + 2 ∗ p n + 2) 2
      using zs by simp
    then have ∀ i<length zs. zs ! i = 2
      using nth-replicate by (metis length-replicate)
    then show ?thesis
      using G by simp
  qed
  show tps13 = tps12
    [2 + 1 := (⌊snd (exc thalt) :#: 0⌋N, 1),
     2 + 2 := (⌊map (λt. snd (exc t) :#: 0) [0..<Suc thalt]⌋NL, 1),
     2 + 4 := (⌊snd (exc thalt) :#: 1⌋N, 1),
     2 + 5 := (⌊map (λt. snd (exc t) :#: 1) [0..<Suc thalt]⌋NL, 1),
     2 + 7 := exc thalt <!> 0, 2 + 8 := exc thalt <!> 1]
    unfolding tps13-def tps12-def by (simp add: list-update-swap[of - 9])
  show tps12 ! 2 = ⌈▷⌉
    using tps12-def tps0 onesie-1 by simp
  show tps12 ! (2 + 1) = (⌊0⌋N, 0)
    using tps12-def tps0 canrepr-0 by simp
  show tps12 ! (2 + 2) = (⌊[]⌋NL, 0)
    using tps12-def tps0 nlcontents-Nil by simp
  show tps12 ! (2 + 3) = ⌈▷⌉
    using tps12-def tps0 onesie-1 by simp
  show tps12 ! (2 + 4) = (⌊0⌋N, 0)
    using tps12-def tps0 canrepr-0 by simp
  show tps12 ! (2 + 5) = (⌊[]⌋NL, 0)
    using tps12-def tps0 nlcontents-Nil by simp
  show tps12 ! (2 + 6) = ⌈▷⌉
    using tps12-def tps0 onesie-1 by simp
  show tps12 ! (2 + 7) = (⌊zs⌋, 0)
    using tps12-def k tps0 by simp
  show tps12 ! (2 + 8) = (⌊[]⌋, 0)
    using tps12-def tps0 by simp
  show ttt = 82 + 11 ∗ n^2 + (d-polynomial p + d-polynomial p ∗ (nlength n)^2) +
      3 ∗ nlength (p n) + 3 ∗ max (nlength n) (nlength (p n)) + 2 ∗ nlength (n + p n) +
      2 ∗ nlength (Suc (n + p n)) + 7 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n))) +
      (2 ∗ n + 2 ∗ p n) ∗ (12 + 2 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n)))) +
      27 ∗ Suc thalt ∗ (9 + 2 ∗ nlength thalt)
    using assms by simp
qed

definition tpsA ≡ tps0
  [9 := exc thalt <!> 0,
   3 := (⌊exc thalt <#> 0⌋N, 1),
   6 := (⌊exc thalt <#> 1⌋N, 1),
   10 := exc thalt <!> 1]
```

```
definition tps14 ≡ tps0
  [9 := exc thalt <!> 0,
   11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊2 ∗ Suc (n + p n)⌋_N, 1),
   17 := (⌊Suc thalt⌋_N, 1),
   3 := (⌊exc thalt <#> 0⌋_N, 1),
   4 := (⌊map (λt. exc t <#> 0) [0..<Suc thalt]⌋_NL, 1),
   6 := (⌊exc thalt <#> 1⌋_N, 1),
   7 := (⌊map (λt. exc t <#> 1) [0..<Suc thalt]⌋_NL, 1),
   10 := exc thalt <!> 1]

lemma tm14:
  assumes ttt = 87 + 11 ∗ n^2 + (d-polynomial p + d-polynomial p ∗ (nlength n)^2) +
    3 ∗ nlength (p n) + 3 ∗ max (nlength n) (nlength (p n)) + 2 ∗ nlength (n + p n) +
    2 ∗ nlength (Suc (n + p n)) + 7 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n))) +
    (2 ∗ n + 2 ∗ p n) ∗ (12 + 2 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n)))) +
    (27 + 27 ∗ thalt) ∗ (9 + 2 ∗ nlength thalt) +
    14 ∗ (nllength (map (λt. exc t <#> 0) [0..<thalt] @ [exc thalt <#> 0]))^2
  shows transforms tm14 tps0 ttt tps14
  unfolding tm14-def
proof (tform)
  show 4 < length tps13 17 < length tps13
    using tps13-def k by (simp-all only: length-list-update)
  show tps13 ! 4 = (⌊map (λt. exc t <#> 0) [0..<Suc thalt]⌋_NL, 1)
    using tps13-def k by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
  show tps13 ! 17 = (⌊0⌋_N, 1)
    using tps13-def k by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
  show tps14 = tps13
      [17 := (⌊length (map (λt. snd (exc t) :#: 0) [0..<Suc thalt])⌋_N, 1)]
    unfolding tps14-def tps13-def by (simp add: list-update-swap[of 17])
  show ttt = 82 + 11 ∗ n^2 + (d-polynomial p + d-polynomial p ∗ (nlength n)^2) +
      3 ∗ nlength (p n) +
      3 ∗ max (nlength n) (nlength (p n)) +
      2 ∗ nlength (n + p n) +
      2 ∗ nlength (Suc (n + p n)) +
      7 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n))) +
      (2 ∗ n + 2 ∗ p n) ∗ (12 + 2 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n)))) +
      (27 + 27 ∗ thalt) ∗ (9 + 2 ∗ nlength thalt) +
      (14 ∗ (nllength (map (λt. snd (exc t) :#: 0) [0..<Suc thalt]))^2 + 5)
    using assms by simp
qed

definition tps14' ≡ tpsA
  [11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊2 ∗ Suc (n + p n)⌋_N, 1),
   17 := (⌊Suc thalt⌋_N, 1),
   4 := (⌊map (λt. exc t <#> 0) [0..<Suc thalt]⌋_NL, 1),
   7 := (⌊map (λt. exc t <#> 1) [0..<Suc thalt]⌋_NL, 1)]

lemma tps14': tps14' = tps14
  unfolding tps14'-def tps14-def tpsA-def by (simp add: list-update-swap)

lemma len-tpsA: length tpsA = k
  using tpsA-def k by simp

lemma tm14' [transforms-intros]:
  assumes ttt = 87 + 11 ∗ n^2 + (d-polynomial p + d-polynomial p ∗ (nlength n)^2) +
    3 ∗ nlength (p n) + 3 ∗ max (nlength n) (nlength (p n)) + 2 ∗ nlength (n + p n) +
    2 ∗ nlength (Suc (n + p n)) + 7 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n))) +
    (2 ∗ n + 2 ∗ p n) ∗ (12 + 2 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n)))) +
    (27 + 27 ∗ thalt) ∗ (9 + 2 ∗ nlength thalt) +
```

14 ∗ (nllength (map (λt. exc t <#> 0) [0..<thalt] @ [exc thalt <#> 0]))^2
shows *transforms tm14 tps0 ttt tps14'*
using *tm14 tps14' assms* **by** *simp*

definition *tps15* ≡ *tpsA*
[11 := (⌊n⌋$_N$, 1),
15 := (⌊p n⌋$_N$, 1),
16 := (⌊2 ∗ Suc (n + p n)⌋$_N$, 1),
17 := (⌊thalt⌋$_N$, 1),
4 := (⌊map (λt. exc t <#> 0) [0..<Suc thalt]⌋$_{NL}$, 1),
7 := (⌊map (λt. exc t <#> 1) [0..<Suc thalt]⌋$_{NL}$, 1)]

lemma *tm15* [*transforms-intros*]:
assumes *ttt* = 95 + 11 ∗ n^2 + (d-polynomial p + d-polynomial p ∗ (nlength n)^2) +
3 ∗ nlength (p n) + 3 ∗ max (nlength n) (nlength (p n)) + 2 ∗ nlength (n + p n) +
2 ∗ nlength (Suc (n + p n)) + 7 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n))) +
(2 ∗ n + 2 ∗ p n) ∗ (12 + 2 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n)))) +
(27 + 27 ∗ thalt) ∗ (9 + 2 ∗ nlength thalt) +
14 ∗ (nllength (map (λt. exc t <#> 0) [0..<thalt] @ [exc thalt <#> 0]))^2 +
2 ∗ nlength (Suc thalt)
shows *transforms tm15 tps0 ttt tps15*
unfolding *tm15-def*
proof (*tform tps*: *tps14'-def len-tpsA k time*: *assms*)
show *tps15* = *tps14'*[17 := (⌊Suc thalt − 1⌋$_N$, 1)]
unfolding *tps15-def tps14'-def* **by** (*simp add*: *list-update-swap*)
qed

definition *tps16* ≡ *tpsA*
[11 := (⌊n⌋$_N$, 1),
15 := (⌊p n⌋$_N$, 1),
16 := (⌊2 ∗ Suc (n + p n)⌋$_N$, 1),
17 := (⌊thalt⌋$_N$, 1),
4 := (⌊map (λt. exc t <#> 0) [0..<Suc thalt]⌋$_{NL}$, 1),
7 := (⌊map (λt. exc t <#> 1) [0..<Suc thalt]⌋$_{NL}$, 1),
18 := (⌊2 ∗ Suc (n + p n)⌋$_N$, 1)]

lemma *tm16* [*transforms-intros*]:
assumes *ttt* = 109 + 11 ∗ n^2 + (d-polynomial p + d-polynomial p ∗ (nlength n)^2) +
3 ∗ nlength (p n) + 3 ∗ max (nlength n) (nlength (p n)) + 2 ∗ nlength (n + p n) +
2 ∗ nlength (Suc (n + p n)) + 10 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n))) +
(2 ∗ n + 2 ∗ p n) ∗ (12 + 2 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n)))) +
(27 + 27 ∗ thalt) ∗ (9 + 2 ∗ nlength thalt) +
14 ∗ (nllength (map (λt. exc t <#> 0) [0..<thalt] @ [exc thalt <#> 0]))^2 +
2 ∗ nlength (Suc thalt)
shows *transforms tm16 tps0 ttt tps16*
unfolding *tm16-def*
proof (*tform tps*: *tps15-def tps16-def k len-tpsA*)
have *tps15* ! 18 = *tpsA* ! 18
using *tps15-def* **by** *simp*
also have ... = *tps0* ! 18
using *tpsA-def* **by** *simp*
also have ... = (⌊0⌋$_N$, 1)
using *tps0 canrepr-0 k* **by** *simp*
finally show *tps15* ! 18 = (⌊0⌋$_N$, 1) **.**
show *ttt* = 95 + 11 ∗ n^2 + (d-polynomial p + d-polynomial p ∗ (nlength n)^2) +
3 ∗ nlength (p n) + 3 ∗ max (nlength n) (nlength (p n)) +
2 ∗ nlength (n + p n) + 2 ∗ nlength (Suc (n + p n)) +
7 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n))) +
(2 ∗ n + 2 ∗ p n) ∗ (12 + 2 ∗ nlength (Suc (Suc (2 ∗ n + 2 ∗ p n)))) + (27 + 27 ∗ thalt) ∗ (9 + 2 ∗ nlength thalt) +
14 ∗ (nllength
(map (λt. snd (exc t) :#: 0) [0..<thalt] @ [snd (exc thalt) :#: 0]))^2 +
2 ∗ nlength (Suc thalt) + (14 + 3 ∗ (nlength (Suc (Suc (2 ∗ n + 2 ∗ p n))) + nlength 0))

using *assms* **by** *simp*
qed

definition *tps17* ≡ *tpsA*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*2* ∗ *Suc* (*n* + *p n*)⌋$_N$, *1*),
17 := (⌊*thalt*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc t* <#> *0*) [*0*..<*Suc thalt*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc t* <#> *1*) [*0*..<*Suc thalt*]⌋$_{NL}$, *1*),
18 := (⌊*Suc* (*2* ∗ *Suc* (*n* + *p n*))⌋$_N$, *1*)]

lemma *tm17* [*transforms-intros*]:
assumes *ttt* = *114* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) +
3 ∗ *nlength* (*p n*) + *3* ∗ *max* (*nlength n*) (*nlength* (*p n*)) + *2* ∗ *nlength* (*n* + *p n*) +
2 ∗ *nlength* (*Suc* (*n* + *p n*)) + *10* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
(*2* ∗ *n* + *2* ∗ *p n*) ∗ (*12* + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))) +
(*27* + *27* ∗ *thalt*) ∗ (*9* + *2* ∗ *nlength thalt*) +
14 ∗ (*nllength* (*map* (λ*t*. *exc t* <#> *0*) [*0*..<*thalt*] @ [*exc thalt* <#> *0*]))^2 +
2 ∗ *nlength* (*Suc thalt*) + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))
shows *transforms tm17 tps0 ttt tps17*
unfolding *tm17-def* **by** (*tform tps*: *tps16-def tps17-def k len-tpsA time*: *assms*)

definition *tps18* ≡ *tpsA*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*2* ∗ *Suc* (*n* + *p n*)⌋$_N$, *1*),
17 := (⌊*thalt*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc t* <#> *0*) [*0*..<*Suc thalt*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc t* <#> *1*) [*0*..<*Suc thalt*]⌋$_{NL}$, *1*),
18 := (⌊*thalt* + *Suc* (*2* ∗ *Suc* (*n* + *p n*))⌋$_N$, *1*)]

lemma *tm18* [*transforms-intros*]:
assumes *ttt* = *124* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) +
3 ∗ *nlength* (*p n*) + *3* ∗ *max* (*nlength n*) (*nlength* (*p n*)) + *2* ∗ *nlength* (*n* + *p n*) +
2 ∗ *nlength* (*Suc* (*n* + *p n*)) + *10* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
(*2* ∗ *n* + *2* ∗ *p n*) ∗ (*12* + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))) +
(*27* + *27* ∗ *thalt*) ∗ (*9* + *2* ∗ *nlength thalt*) +
14 ∗ (*nllength* (*map* (λ*t*. *exc t* <#> *0*) [*0*..<*thalt*] @ [*exc thalt* <#> *0*]))^2 +
2 ∗ *nlength* (*Suc thalt*) + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
3 ∗ *max* (*nlength thalt*) (*nlength* (*Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))))
shows *transforms tm18 tps0 ttt tps18*
unfolding *tm18-def* **by** (*tform tps*: *tps17-def tps18-def k len-tpsA time*: *assms*)

definition *tps19* ≡ *tpsA*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*2* ∗ *Suc* (*n* + *p n*)⌋$_N$, *1*),
17 := (⌊*thalt*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc t* <#> *0*) [*0*..<*Suc thalt*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc t* <#> *1*) [*0*..<*Suc thalt*]⌋$_{NL}$, *1*),
18 := (⌊*thalt* + *Suc* (*2* ∗ *Suc* (*n* + *p n*))⌋$_N$, *1*),
19 := (⌊*max G* (*length M*)⌋$_N$, *1*)]

lemma *tm19* [*transforms-intros*]:
assumes *ttt* = *134* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) +
3 ∗ *nlength* (*p n*) + *3* ∗ *max* (*nlength n*) (*nlength* (*p n*)) + *2* ∗ *nlength* (*n* + *p n*) +
2 ∗ *nlength* (*Suc* (*n* + *p n*)) + *10* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
(*2* ∗ *n* + *2* ∗ *p n*) ∗ (*12* + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))) +
(*27* + *27* ∗ *thalt*) ∗ (*9* + *2* ∗ *nlength thalt*) +
14 ∗ (*nllength* (*map* (λ*t*. *exc t* <#> *0*) [*0*..<*thalt*] @ [*exc thalt* <#> *0*]))^2 +
2 ∗ *nlength* (*Suc thalt*) + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
3 ∗ *max* (*nlength thalt*) (*nlength* (*Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))))) +

    2 * nlength (max G (length M))
  shows transforms tm19 tps0 ttt tps19
  unfolding tm19-def
proof (tform tps: assms nlength-0)
  have tps18 ! 19 = tpsA ! 19
    using tps18-def **by** simp
  also have ... = tps0 ! 19
    using tpsA-def **by** simp
  also have ... = ($\lfloor 0 \rfloor_N$, 1)
    using tps0 canrepr-0 k **by** simp
  finally show tps18 ! 19 = ($\lfloor 0 \rfloor_N$, 1) **.**
  show 19 < length tps18
    using tps18-def len-tpsA k **by** simp
  show tps19 = tps18[19 := ($\lfloor$max G (length M)$\rfloor_N$, 1)]
    using tps19-def tps18-def len-tpsA k **by** presburger
qed

definition tps20 ≡ tpsA
  [11 := ($\lfloor n \rfloor_N$, 1),
   15 := ($\lfloor$p n$\rfloor_N$, 1),
   16 := ($\lfloor$2 * Suc (n + p n)$\rfloor_N$, 1),
   17 := ($\lfloor$thalt$\rfloor_N$, 1),
   4 := ($\lfloor$map (λt. exc t <#> 0) [0..<Suc thalt]$\rfloor_{NL}$, 1),
   7 := ($\lfloor$map (λt. exc t <#> 1) [0..<Suc thalt]$\rfloor_{NL}$, 1),
   18 := ($\lfloor$thalt + Suc (2 * Suc (n + p n))$\rfloor_N$, 1),
   19 := ($\lfloor$max G (length M)$\rfloor_N$, 1),
   20 := ($\lfloor$(thalt + Suc (2 * Suc (n + p n))) * max G (length M)$\rfloor_N$, 1)]

lemma tm20:
  assumes ttt = 138 + 11 * n^2 + (d-polynomial p + d-polynomial p * (nlength n)^2) +
    3 * nlength (p n) + 3 * max (nlength n) (nlength (p n)) + 2 * nlength (n + p n) +
    2 * nlength (Suc (n + p n)) + 10 * nlength (Suc (Suc (2 * n + 2 * p n))) +
    (2 * n + 2 * p n) * (12 + 2 * nlength (Suc (Suc (2 * n + 2 * p n)))) +
    (27 + 27 * thalt) * (9 + 2 * nlength thalt) +
    14 * (nllength (map (λt. exc t <#> 0) [0..<thalt] @ [exc thalt <#> 0]))^2 +
    2 * nlength (Suc thalt) + 2 * nlength (Suc (Suc (2 * n + 2 * p n))) +
    3 * max (nlength thalt) (nlength (Suc (Suc (Suc (2 * n + 2 * p n))))) +
    2 * nlength (max G (length M)) +
    26 * (nlength (Suc (Suc (Suc (thalt + (2 * n + 2 * p n))))) + nlength (max G (length M))) *
     (nlength (Suc (Suc (Suc (thalt + (2 * n + 2 * p n))))) + nlength (max G (length M)))
  shows transforms tm20 tps0 ttt tps20
  unfolding tm20-def
proof (tform time: assms)
  have tps19 ! 20 = tpsA ! 20
    using tps19-def **by** simp
  also have ... = tps0 ! 20
    using tpsA-def **by** simp
  also have ... = ($\lfloor 0 \rfloor_N$, 1)
    using tps0 canrepr-0 k **by** simp
  finally show tps19 ! 20 = ($\lfloor 0 \rfloor_N$, 1) **.**
  show tps20 = tps19
      [20 := ($\lfloor$Suc (Suc (Suc (thalt + (2 * n + 2 * p n)))) * max G (length M)$\rfloor_N$, 1)]
    unfolding tps20-def tps19-def **by** (simp add: list-update-swap)
  show 18 < length tps19 19 < length tps19 20 < length tps19
    using tps19-def k len-tpsA **by** (simp-all only: length-list-update)
  have tps19 ! 18 = ($\lfloor$thalt + Suc (2 * Suc (n + p n))$\rfloor_N$, 1)
   using tps19-def tpsA-def len-tpsA k tps0 **by** (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  then show tps19 ! 18 = ($\lfloor$Suc (Suc (Suc (thalt + (2 * n + 2 * p n))))$\rfloor_N$, 1)
    by simp
  show tps19 ! 19 = ($\lfloor$max G (length M)$\rfloor_N$, 1)
   using tps19-def tpsA-def len-tpsA k tps0 **by** (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
qed

lemma *tm20'* [*transforms-intros*]:
assumes *ttt* = (*2* ∗ *d-polynomial p* + *826*) ∗ (*max G* (*length M*) + *thalt* + *Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))) ^ *4*
shows *transforms tm20 tps0 ttt tps20*
proof −
let *?ttt* = *138* + *11* ∗ n^2 + (*d-polynomial p* + *d-polynomial p* ∗ (*nlength n*)^2) +
3 ∗ *nlength* (*p n*) + *3* ∗ *max* (*nlength n*) (*nlength* (*p n*)) + *2* ∗ *nlength* (*n* + *p n*) +
2 ∗ *nlength* (*Suc* (*n* + *p n*)) + *10* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
(*2* ∗ *n* + *2* ∗ *p n*) ∗ (*12* + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))) +
(*27* + *27* ∗ *thalt*) ∗ (*9* + *2* ∗ *nlength thalt*) +
14 ∗ (*nllength* (*map* (λ*t*. *exc t* <#> *0*) [*0*..<*thalt*] @ [*exc thalt* <#> *0*]))^2 +
2 ∗ *nlength* (*Suc thalt*) + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
3 ∗ *max* (*nlength thalt*) (*nlength* (*Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))))) +
2 ∗ *nlength* (*max G* (*length M*)) +
26 ∗ (*nlength* (*Suc* (*Suc* (*Suc* (*thalt* + (*2* ∗ *n* + *2* ∗ *p n*))))) + *nlength* (*max G* (*length M*))) ∗
(*nlength* (*Suc* (*Suc* (*Suc* (*thalt* + (*2* ∗ *n* + *2* ∗ *p n*))))) + *nlength* (*max G* (*length M*)))
let *?a* = *3* ∗ *nlength* (*p n*) + *3* ∗ *max* (*nlength n*) (*nlength* (*p n*)) + *2* ∗ *nlength* (*n* + *p n*) +
2 ∗ *nlength* (*Suc* (*n* + *p n*)) + *10* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
(*2* ∗ *n* + *2* ∗ *p n*) ∗ (*12* + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))) + (*27* + *27* ∗ *thalt*) ∗ (*9* + *2* ∗ *nlength thalt*)
let *?b* = *2* ∗ *nlength* (*Suc thalt*) + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
3 ∗ *max* (*nlength thalt*) (*nlength* (*Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))))) +
2 ∗ *nlength* (*max G* (*length M*)) +
26 ∗ (*nlength* (*Suc* (*Suc* (*Suc* (*thalt* + (*2* ∗ *n* + *2* ∗ *p n*))))) + *nlength* (*max G* (*length M*))) ∗
(*nlength* (*Suc* (*Suc* (*Suc* (*thalt* + (*2* ∗ *n* + *2* ∗ *p n*))))) + *nlength* (*max G* (*length M*)))
let *?m* = *max G* (*length M*) + *thalt* + *Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))
define *m* **where** *m* = *max G* (*length M*) + *thalt* + *Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))
note *m-def* [*simp*]
have ∗∗: *y* ≤ *y* ∗ *m* **for** *y*
by *simp*
have ∗: *nlength y* ≤ *m* **if** *y* ≤ *m* **for** *y*
using *nlength-mono*[*OF that*] *nlength-mono* **by** (*meson dual-order.trans nlength-le-n*)
have *1*: *nlength* (*p n*) ≤ *m*
using ∗ **by** *simp*
have *2*: *max* (*nlength n*) (*nlength* (*p n*)) ≤ *m*
using ∗ **by** *simp*
have *3*: *nlength* (*n* + *p n*) ≤ *m*
using ∗ **by** *simp*
have *4*: *nlength* (*Suc* (*n* + *p n*)) ≤ *m*
using ∗ **by** *simp*
have *5*: *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) ≤ *m*
using ∗ **by** *simp*
have *6*: *nlength n* ≤ *m*
using ∗ **by** *simp*
have *7*: *2* ∗ *n* + *2* ∗ *p n* ≤ *m*
by *simp*
have *8*: *thalt* ≤ *m nlength thalt* ≤ *m nlength* (*Suc thalt*) ≤ *m*
using ∗ **by** *simp-all*
have *10*: *max* (*nlength thalt*) (*nlength* (*Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))))) ≤ *m*
using ∗ **by** *simp*
have *11*: *nlength* (*Suc* (*Suc* (*Suc* (*thalt* + (*2* ∗ *n* + *2* ∗ *p n*))))) ≤ *m*
using ∗ **by** *simp*
have *12*: *nlength* (*Suc* (*Suc* (*Suc* (*thalt* + (*2* ∗ *n* + *2* ∗ *p n*))))) + *nlength* (*max G* (*length M*)) ≤ *m*
using ∗ *nlength-le-n* **by** (*smt* (*verit*) *ab-semigroup-add-class.add-ac*(*1*) *add.commute add-Suc-right add-le-mono m-def*)
have *13*: *nlength* (*max G* (*length M*)) ≤ *m*
using *12* **by** *simp*
have *14*: *Suc* (*nlength thalt*) ≤ *m*
proof −
have *nlength thalt* ≤ *nlength m*
using *nlength-mono* **by** *simp*
moreover have *m* ≥ *3*
by *simp*

ultimately have *nlength thalt* < *m*
using *nlength-less-n dual-order.strict-trans2* **by** *blast*
then show *?thesis*
by *simp*
qed
have *15*: *Suc thalt* ≤ *m*
by *simp*

have *?a* ≤ *20* ∗ *m* +
(*2* ∗ *n* + *2* ∗ *p n*) ∗ (*12* + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))) + (*27* + *27* ∗ *thalt*) ∗ (*9* + *2* ∗ *nlength thalt*)
using *1 2 3 4 5* **by** *linarith*
also have ... ≤ *20* ∗ *m* + *m* ∗ (*12* + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))) + (*27* + *27* ∗ *thalt*) ∗ (*9* + *2* ∗ *nlength thalt*)
using *7* **by** (*metis add.commute add-mono-thms-linordered-semiring*(*2*) *mult-Suc-right mult-le-cancel2*)
also have ... ≤ *20* ∗ *m* + *m* ∗ (*12* + *2* ∗ *m*) + (*27* + *27* ∗ *thalt*) ∗ (*9* + *2* ∗ *nlength thalt*)
using *5* **by** (*meson add-left-mono add-mono-thms-linordered-semiring*(*3*) *mult-le-mono2*)
also have ... ≤ *20* ∗ *m* + *m* ∗ (*12* + *2* ∗ *m*) + (*27* + *27* ∗ *m*) ∗ (*9* + *2* ∗ *m*)
using *8 add-le-mono le-refl mult-le-mono* **by** *presburger*
also have ... ≤ *20* ∗ *m* + *m* ∗ (*12* ∗ *m* + *2* ∗ *m*) + (*27* ∗ *m* + *27* ∗ *m*) ∗ (*9* + *2* ∗ *m*)
using ∗∗ **by** (*meson add-le-mono add-mono-thms-linordered-semiring*(*2*) *add-mono-thms-linordered-semiring*(*3*) *mult-le-mono1 mult-le-mono2*)
also have ... ≤ *20* ∗ *m* + *m* ∗ (*12* ∗ *m* + *2* ∗ *m*) + (*27* ∗ *m* + *27* ∗ *m*) ∗ (*9* ∗ *m* + *2* ∗ *m*)
using ∗∗ **by** *simp*
also have ... = *20* ∗ *m* + *m* ∗ *14* ∗ *m* + *54* ∗ *m* ∗ *11* ∗ *m*
by *algebra*
also have ... = *20* ∗ *m* + *14* ∗ *m* ^ *2* + *594* ∗ *m* ^ *2*
by *algebra*
also have ... = *20* ∗ *m* + *608* ∗ *m* ^ *2*
by *simp*
also have ... ≤ *20* ∗ *m* ^ *2* + *608* ∗ *m* ^ *2*
using *linear-le-pow* **by** (*meson add-le-mono1 mult-le-mono2 zero-less-numeral*)
also have ... = *628* ∗ *m* ^ *2*
by *simp*
finally have *part1*: *?a* ≤ *628* ∗ *m* ^ *2* **.**

have *nllength* (*map* (λ*t*. *exc t* <#> *0*) [*0*..<*thalt*] @ [*exc thalt* <#> *0*]) ≤ *Suc* (*nlength thalt*) ∗ *Suc thalt*
proof −
have *exc t* <#> *0* ≤ *thalt* **if** *t* ≤ *thalt* **for** *t*
using *that M-tm head-pos-le-halting-time thalt*(*2*) *zero-less-numeral* **by** *blast*
then have *y* ≤ *thalt* **if** *y* ∈ *set* (*map* (λ*t*. *exc t* <#> *0*) [*0*..<*Suc thalt*]) **for** *y*
using *that* **by** *force*
then have *nllength* (*map* (λ*t*. *exc t* <#> *0*) [*0*..<*Suc thalt*]) ≤ *Suc* (*nlength thalt*) ∗ *Suc thalt*
(**is** *nllength ?ns* ≤ -)
using *nllength-le-len-mult-max*[*of ?ns thalt*] **by** *simp*
then show *?thesis*
by *simp*
qed
then have *part2*: *nllength* (*map* (λ*t*. *exc t* <#> *0*) [*0*..<*thalt*] @ [*exc thalt* <#> *0*]) ≤ *m* ∗ *m*
using *14 15* **by** (*meson le-trans mult-le-mono*)

have *?b* = *2* ∗ *nlength* (*Suc thalt*) + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
3 ∗ *max* (*nlength thalt*) (*nlength* (*Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))))) + *2* ∗ *nlength* (*max G* (*length M*)) +
26 ∗ (*nlength* (*Suc* (*Suc* (*Suc* (*thalt* + (*2* ∗ *n* + *2* ∗ *p n*))))) + *nlength* (*max G* (*length M*))) ^ *2*
by *algebra*
also have ... ≤ *2* ∗ *nlength* (*Suc thalt*) + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
3 ∗ *max* (*nlength thalt*) (*nlength* (*Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))))) + *2* ∗ *nlength* (*max G* (*length M*)) +
26 ∗ *m* ^ *2*
using *12* **by** *simp*
also have ... ≤ *2* ∗ *nlength* (*Suc thalt*) + *2* ∗ *nlength* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*))) +
3 ∗ *m* + *2* ∗ *nlength* (*max G* (*length M*)) + *26* ∗ *m* ^ *2*
using *10* **by** *linarith*
also have ... ≤ *2* ∗ *m* + *2* ∗ *m* + *3* ∗ *m* + *2* ∗ *m* + *26* ∗ *m* ^ *2*

```
    using 13 8 5 by simp
  also have ... = 9 * m + 26 * m ^ 2
    by simp
  also have ... ≤ 9 * m ^ 2 + 26 * m ^ 2
    using linear-le-pow by (meson add-le-mono1 mult-le-mono2 zero-less-numeral)
  also have ... = 35 * m ^ 2
    by simp
  finally have part3: ?b ≤ 35 * m ^ 2 .

  have ?ttt = 138 + 11 * n² + (d-polynomial p + d-polynomial p * (nlength n)²) + ?a +
      14 * (nllength (map (λt. exc t <#> 0) [0..<thalt] @ [exc thalt <#> 0]))² + ?b
    by simp
  also have ... ≤ 138 + 11 * n² + d-polynomial p + d-polynomial p * (nlength n)² + ?a + 14 * (m * m)² + ?b
    using part2 by simp
  also have ... ≤ 138 + 11 * n² + d-polynomial p + d-polynomial p * (nlength n)² + ?a + 14 * (m * m)² +
35 * m ^ 2
    using part3 by linarith
  also have ... ≤ 138 + 11 * n² + d-polynomial p + d-polynomial p * (nlength n)² + 628 * m ^ 2 + 14 * (m
* m)² + 35 * m ^ 2
    using part1 by linarith
  also have ... = 138 + 11 * n² + d-polynomial p + d-polynomial p * (nlength n)² + 663 * m ^ 2 + 14 * m ^
4
    by algebra
  also have ... ≤ 138 + 11 * m ^ 2 + d-polynomial p + d-polynomial p * (nlength n)² + 663 * m ^ 2 + 14 *
m ^ 4
    by simp
  also have ... ≤ 138 + 11 * m ^ 2 + d-polynomial p + d-polynomial p * m ^ 2 + 663 * m ^ 2 + 14 * m ^ 4
    using 6 by simp
  also have ... = 138 + d-polynomial p + d-polynomial p * m ^ 2 + 674 * m ^ 2 + 14 * m ^ 4
    by simp
  also have ... ≤ 138 + d-polynomial p + d-polynomial p * m ^ 4 + 674 * m ^ 2 + 14 * m ^ 4
    using pow-mono'[of 2 4] by simp
  also have ... ≤ 138 + d-polynomial p + d-polynomial p * m ^ 4 + 674 * m ^ 4 + 14 * m ^ 4
    using pow-mono'[of 2 4] by simp
  also have ... = 138 + d-polynomial p + d-polynomial p * m ^ 4 + 688 * m ^ 4
    by simp
  also have ... ≤ 138 * m + d-polynomial p + d-polynomial p * m ^ 4 + 688 * m ^ 4
    using ** by simp
  also have ... ≤ 138 * m ^ 4 + d-polynomial p + d-polynomial p * m ^ 4 + 688 * m ^ 4
    using linear-le-pow[of 4 m] by simp
  also have ... = d-polynomial p + d-polynomial p * m ^ 4 + 826 * m ^ 4
    by simp
  also have ... ≤ d-polynomial p * m + d-polynomial p * m ^ 4 + 826 * m ^ 4
    using ** by simp
  also have ... ≤ d-polynomial p * m ^ 4 + d-polynomial p * m ^ 4 + 826 * m ^ 4
    using linear-le-pow[of 4 m] by (simp del: m-def)
  also have ... = 2 * d-polynomial p * m ^ 4 + 826 * m ^ 4
    by simp
  also have ... = (2 * d-polynomial p + 826) * m ^ 4
    by algebra
  finally have ?ttt ≤ (2 * d-polynomial p + 826) * m ^ 4 .
  then have ?ttt ≤ ttt
    using assms by simp
  then show ?thesis
    using tm20 transforms-monotone by fast
qed

end

end

lemma transforms-tm-PHI-initI:
  fixes G :: nat and M :: machine and p :: nat ⇒ nat
```

fixes *k H thalt* :: *nat* **and** *tps tps'* :: *tape list* **and** *xs zs* :: *symbol list*
assumes *poly-p*: *polynomial p*
and *M-tm*: *turing-machine 2 G M*
and *k*: $k = length\ tps\ 20 < k$
and *H*: $H = max\ G\ (length\ M)$
and *xs*: *bit-symbols xs*
and *zs*: $zs = 2\ \#\ 2\ \#\ replicate\ (2 * length\ xs + 2 * p\ (length\ xs))\ 2$
assumes *thalt*:
$\forall t{<}thalt.\ fst\ (execute\ M\ (start\text{-}config\ 2\ zs)\ t) < length\ M$
$fst\ (execute\ M\ (start\text{-}config\ 2\ zs)\ thalt) = length\ M$
assumes *tps0*:
$tps\ !\ 0 = (\lfloor xs \rfloor,\ 1)$
$\bigwedge i.\ 0 < i \Longrightarrow i \le 10 \Longrightarrow tps\ !\ i = (\lfloor [] \rfloor,\ 0)$
$\bigwedge i.\ 10 < i \Longrightarrow i < k \Longrightarrow tps\ !\ i = (\lfloor [] \rfloor,\ 1)$
assumes $ttt = (2 * d\text{-}polynomial\ p + 826) * (max\ G\ (length\ M) + thalt + Suc\ (Suc\ (Suc\ (2 * (length\ xs) + 2 * p\ (length\ xs))))) \ \hat{}\ 4$
assumes $tps' = tps$
$[9 := execute\ M\ (start\text{-}config\ 2\ zs)\ thalt <!> 0,$
$3 := (\lfloor execute\ M\ (start\text{-}config\ 2\ zs)\ thalt <\#> 0 \rfloor_N,\ 1),$
$6 := (\lfloor execute\ M\ (start\text{-}config\ 2\ zs)\ thalt <\#> 1 \rfloor_N,\ 1),$
$10 := execute\ M\ (start\text{-}config\ 2\ zs)\ thalt <!> 1,$
$11 := (\lfloor length\ xs \rfloor_N,\ 1),$
$15 := (\lfloor p\ (length\ xs) \rfloor_N,\ 1),$
$16 := (\lfloor 2 * Suc\ ((length\ xs) + p\ (length\ xs)) \rfloor_N,\ 1),$
$17 := (\lfloor thalt \rfloor_N,\ 1),$
$4 := (\lfloor map\ (\lambda t.\ execute\ M\ (start\text{-}config\ 2\ zs)\ t <\#> 0)\ [0..{<}Suc\ thalt] \rfloor_{NL},\ 1),$
$7 := (\lfloor map\ (\lambda t.\ execute\ M\ (start\text{-}config\ 2\ zs)\ t <\#> 1)\ [0..{<}Suc\ thalt] \rfloor_{NL},\ 1),$
$18 := (\lfloor thalt + Suc\ (2 * Suc\ ((length\ xs) + p\ (length\ xs))) \rfloor_N,\ 1),$
$19 := (\lfloor max\ G\ (length\ M) \rfloor_N,\ 1),$
$20 := (\lfloor (thalt + Suc\ (2 * Suc\ ((length\ xs) + p\ (length\ xs)))) * max\ G\ (length\ M) \rfloor_N,\ 1)]$
shows *transforms* (*tm-PHI-init G M p*) *tps ttt tps'*
proof −
interpret *loc*: *turing-machine-PHI-init G M p* **.**
note *ctx* = *poly-p M-tm k H xs zs thalt tps0*
have *transforms loc.tm20 tps ttt* (*loc.tps20 thalt tps xs zs*)
using *assms loc.tm20'*[*OF ctx*] *loc.tps20-def*[*OF ctx*] *loc.tpsA-def*[*OF ctx*]
by *blast*
then have *transforms* (*tm-PHI-init G M p*) *tps ttt* (*loc.tps20 thalt tps xs zs*)
using *loc.tm20-eq-tm-PHI-init* **by** *simp*
moreover have *loc.tps20 thalt tps xs zs* = *tps'*
using *assms loc.tps20-def*[*OF ctx*] *loc.tpsA-def*[*OF ctx*] **by** *presburger*
ultimately show *?thesis*
by *simp*
qed

Next we transfer the semantics of *tm-PHI-init* to the locale *reduction-sat-x*.

lemma (**in** *reduction-sat-x*) *tm-PHI-init* [*transforms-intros*]:
fixes *k* :: *nat* **and** *tps tps'* :: *tape list*
assumes $k = length\ tps$ **and** $20 < k$
assumes
$tps\ !\ 0 = (\lfloor xs \rfloor,\ 1)$
$\bigwedge i.\ 0 < i \Longrightarrow i \le 10 \Longrightarrow tps\ !\ i = (\lfloor [] \rfloor,\ 0)$
$\bigwedge i.\ 10 < i \Longrightarrow i < k \Longrightarrow tps\ !\ i = (\lfloor [] \rfloor,\ 1)$
assumes $ttt = (2 * d\text{-}polynomial\ p + 826) * (H + T' + Suc\ (Suc\ (Suc\ (2 * n + 2 * p\ n)))) \ \hat{}\ 4$
assumes $tps' = tps$
$[9 := exc\ (zeros\ m)\ T' <!> 0,$
$3 := (\lfloor exc\ (zeros\ m)\ T' <\#> 0 \rfloor_N,\ 1),$
$6 := (\lfloor exc\ (zeros\ m)\ T' <\#> 1 \rfloor_N,\ 1),$
$10 := exc\ (zeros\ m)\ T' <!> 1,$
$11 := (\lfloor n \rfloor_N,\ 1),$
$15 := (\lfloor p\ n \rfloor_N,\ 1),$
$16 := (\lfloor 2 * Suc\ (n + p\ n) \rfloor_N,\ 1),$
$17 := (\lfloor T' \rfloor_N,\ 1),$

```
    4 := (⌊map (λt. exc (zeros m) t <#> 0) [0..<Suc T']⌋NL, 1),
    7 := (⌊map (λt. exc (zeros m) t <#> 1) [0..<Suc T']⌋NL, 1),
    18 := (⌊T' + Suc (2 * Suc (n + p n))⌋N, 1),
    19 := (⌊H⌋N, 1),
    20 := (⌊(T' + Suc (2 * Suc (n + p n))) * H⌋N, 1)]
  shows transforms (tm-PHI-init G M p) tps ttt tps'
proof −
  have nx: n = length xs
    by simp
  then have zeros-zs: zeros m = 2 # 2 # replicate (2 * length xs + 2 * p (length xs)) 2
    using zeros m-def by simp
  then have thalt:
      ∀t<T'. fst (exc (zeros m) t) < length M
      fst (exc (zeros m) T') = length M
    using less-TT TT T'-def by metis+
  have H: H = max G (length M)
    using H-def by simp
  have ttt: ttt = (2 * d-polynomial p + 826) *
      (max G (length M) + T' + Suc (Suc (Suc (2 * (length xs) + 2 * p (length xs))))) ^ 4
    using H nx assms(6) by simp
  have tps': tps' = tps
      [9 := exc (zeros m) T' <!> 0,
      3 := (⌊snd (exc (zeros m) T') :#: 0⌋N, 1),
      6 := (⌊snd (exc (zeros m) T') :#: 1⌋N, 1),
      10 := exc (zeros m) T' <!> 1, 11 := (⌊length xs⌋N, 1),
      15 := (⌊p (length xs)⌋N, 1),
      16 := (⌊2 * Suc (length xs + p (length xs))⌋N, 1),
      17 := (⌊T'⌋N, 1),
      4 := (⌊map (λt. snd (exc (zeros m) t) :#: 0) [0..<Suc T']⌋NL, 1),
      7 := (⌊map (λt. snd (exc (zeros m) t) :#: 1) [0..<Suc T']⌋NL, 1),
      18 := (⌊T' + Suc (2 * Suc (length xs + p (length xs)))⌋N, 1),
      19 := (⌊max G (length M)⌋N, 1),
      20 := (⌊(T' + Suc (2 * Suc (length xs + p (length xs)))) * max G (length M)⌋N, 1)]
    using H nx assms(7) by presburger
  show transforms (tm-PHI-init G M p) tps ttt tps'
    using transforms-tm-PHI-initI[OF p tm-M assms(1,2) H bs-xs zeros-zs thalt assms(3,4,5) ttt tps'] .
qed
```

8.3 The actual Turing machine computing the reduction

In this section we put everything together to build a Turing machine that given a string x outputs the CNF formula Φ defined in Chapter 6. In principle this is just a sequence of the TMs *tm-PHI-init*, *tm-PHI0*, …, *tm-PHI9*, where *tm-PHI345* occurs once for each of the formulas Φ_3, Φ_4, and Φ_5. All these TMs are linked by TMs that copy values prepared by *tm-PHI-init* to the tapes where the following TM expects them. Also as the very first step the tape heads on tapes 0 and 11 and beyond must be moved one cell to the right to meet *tm-PHI-init*'s expectations.

The TM will have 110 tapes because we just allocate another batch of tapes for every TM computing a Φ_i, rather than cleaning up and reusing tapes.

The Turing machine for computing Φ is to be defined in the locale *reduction-sat*. We save the space to write the TM in closed form.

```
context reduction-sat
begin

definition tm1 ≡ tm-right-many {i. i < 1 ∨ 10 < i}
definition tm2 ≡ tm1 ;; tm-PHI-init G M p

definition tm3 ≡ tm2 ;; tm-copyn 18 21
definition tm4 ≡ tm3 ;; tm-copyn 19 22
definition tm5 ≡ tm4 ;; tm-right 1
definition tm6 ≡ tm5 ;; tm-PHI0 21
```

definition *tm7* ≡ *tm6* ;; *tm-setn 29 H*
definition *tm8* ≡ *tm7* ;; *tm-PHI1 28*

definition *tm9* ≡ *tm8* ;; *tm-copyn 11 35*
definition *tm10* ≡ *tm9* ;; *tm-setn 36 H*
definition *tm11* ≡ *tm10* ;; *tm-PHI2 35*

definition *tm12* ≡ *tm11* ;; *tm-setn 42 1*
definition *tm13* ≡ *tm12* ;; *tm-setn 43 H*
definition *tm14* ≡ *tm13* ;; *tm-setn 44 2*
definition *tm15* ≡ *tm14* ;; *tm-copyn 11 50*
definition *tm16* ≡ *tm15* ;; *tm-times2incr 50*
definition *tm17* ≡ *tm16* ;; *tm-PHI345 2 42*

definition *tm18* ≡ *tm17* ;; *tm-setn 52 H*
definition *tm19* ≡ *tm18* ;; *tm-setn 53 2*
definition *tm20* ≡ *tm19* ;; *tm-copyn 11 51*
definition *tm21* ≡ *tm20* ;; *tm-times2 51*
definition *tm22* ≡ *tm21* ;; *tm-plus-const 3 51*
definition *tm23* ≡ *tm22* ;; *tm-copyn 16 59*
definition *tm24* ≡ *tm23* ;; *tm-incr 59*
definition *tm25* ≡ *tm24* ;; *tm-PHI345 2 51*

definition *tm26* ≡ *tm25* ;; *tm-setn 61 H*
definition *tm27* ≡ *tm26* ;; *tm-copyn 16 60*
definition *tm28* ≡ *tm27* ;; *tm-incr 60*
definition *tm29* ≡ *tm28* ;; *tm-copyn 60 68*
definition *tm30* ≡ *tm29* ;; *tm-add 17 68*
definition *tm31* ≡ *tm30* ;; *tm-PHI345 1 60*

definition *tm32* ≡ *tm31* ;; *tm-setn 69 2*
definition *tm33* ≡ *tm32* ;; *tm-setn 70 H*
definition *tm34* ≡ *tm33* ;; *tm-PHI6 69*

definition *tm35* ≡ *tm34* ;; *tm-copyn 11 77*
definition *tm36* ≡ *tm35* ;; *tm-times2 77*
definition *tm37* ≡ *tm36* ;; *tm-plus-const 4 77*
definition *tm38* ≡ *tm37* ;; *tm-setn 78 H*
definition *tm39* ≡ *tm38* ;; *tm-copyn 15 83*
definition *tm40* ≡ *tm39* ;; *tm-PHI7 77*

definition *tm41* ≡ *tm40* ;; *tm-copyn 18 84*
definition *tm42* ≡ *tm41* ;; *tm-add 17 84*
definition *tm43* ≡ *tm42* ;; *tm-add 17 84*
definition *tm44* ≡ *tm43* ;; *tm-add 17 84*
definition *tm45* ≡ *tm44* ;; *tm-incr 84*
definition *tm46* ≡ *tm45* ;; *tm-setn 85 H*
definition *tm47* ≡ *tm46* ;; *tm-PHI8 84*

definition *tm48* ≡ *tm47* ;; *tm-copyn 20 91*
definition *tm49* ≡ *tm48* ;; *tm-setn 92 H*
definition *tm50* ≡ *tm49* ;; *tm-setn 93 Z*
definition *tm51* ≡ *tm50* ;; *tm-copyn 17 94*
definition *tm52* ≡ *tm51* ;; *tm-set 95* (*numlistlist* (*formula-n* ψ))
definition *tm53* ≡ *tm52* ;; *tm-set 96* (*numlistlist* (*formula-n* ψ'))
definition *tm54* ≡ *tm53* ;; *tm-setn 97 1*
definition *tm55* ≡ *tm54* ;; *tm-PHI9 4 7 91*

definition *tm56* ≡ *tm55* ;; *tm-cr 1*
definition *tm57* ≡ *tm56* ;; *tm-cp-until 1 109* {*0*}
definition *tm58* ≡ *tm57* ;; *tm-erase-cr 1*
definition *tm59* ≡ *tm58* ;; *tm-cr 109*
definition *tm60* ≡ *tm59* ;; *tm-binencode 109 1*

definition H' :: *nat* **where**
$H' \equiv Suc\ (Suc\ H)$

lemma *H-gr-3*: $H > 3$
using *H-def tm-M turing-machine-def* **by** *auto*

lemma H': $H' \geq Suc\ (length\ M)\ H' \geq G\ H' \geq 6$
using H'*-def H-ge-length-M H-ge-G H-gr-3* **by** *simp-all*

lemma *tm40-tm*: *turing-machine 110* H' *tm40*
unfolding *tm40-def tm39-def tm38-def tm37-def tm36-def tm35-def tm34-def tm33-def tm32-def tm31-def*
unfolding *tm30-def tm29-def tm28-def tm27-def tm26-def tm25-def tm24-def tm23-def tm22-def tm21-def*
unfolding *tm20-def tm19-def tm18-def tm17-def tm16-def tm15-def tm14-def tm13-def tm12-def tm11-def*
unfolding *tm10-def tm9-def tm8-def tm7-def tm6-def tm5-def tm4-def tm3-def tm2-def tm1-def*
using H'
tm-copyn-tm tm-add-tm tm-incr-tm tm-times2-tm tm-setn-tm tm-times2incr-tm
tm-plus-const-tm tm-right-tm tm-right-many-tm
tm-PHI-init-tm[*OF tm-M*] *tm-PHI0-tm tm-PHI1-tm tm-PHI2-tm tm-PHI345-tm tm-PHI6-tm tm-PHI7-tm*
by *simp*

lemma *tm55-tm*: *turing-machine 110* H' *tm55*
unfolding *tm55-def tm54-def tm53-def tm52-def tm51-def*
unfolding *tm50-def tm49-def tm48-def tm47-def tm46-def tm45-def tm44-def tm43-def tm42-def tm41-def*
using *tm40-tm* H'
tm-copyn-tm tm-add-tm tm-incr-tm tm-setn-tm tm-set-tm[*OF - - - symbols-lt-numlistlist*]
tm-PHI8-tm tm-PHI9-tm
by *simp*

lemma *tm60-tm*: *turing-machine 110* H' *tm60*
unfolding *tm60-def tm59-def tm58-def tm57-def tm56-def*
using *tm55-tm* H' *tm-erase-cr-tm tm-cr-tm tm-cp-until-tm tm-binencode-tm*
by *simp*

end

Unlike before, we prove the semantics inside locale *reduction-sat-x* since we need not be concerned with "polluting" the namespace of the locale. After all there will not be any more Turing machines.

context *reduction-sat-x*
begin

context
fixes *tps0* :: *tape list*
assumes *k*: *110 = length tps0*
assumes *tps0*:
$tps0\ !\ 0 = (\lfloor xs \rfloor,\ 0)$
$\bigwedge i.\ 0 < i \Longrightarrow i < 110 \Longrightarrow tps0\ !\ i = (\lfloor [] \rfloor,\ 0)$
begin

definition $tps1 \equiv map\ (\lambda j.\ if\ j < 1 \lor 10 < j\ then\ tps0\ !\ j\ |+|\ 1\ else\ tps0\ !\ j)\ [0..<110]$

lemma *lentps1*: $length\ tps1 = 110$
using *tps1-def* **by** *simp*

lemma *tps1*:
$0 < j \Longrightarrow j < 10 \Longrightarrow tps1\ !\ j = (\lfloor [] \rfloor,\ 0)$
$10 < j \Longrightarrow j < 110 \Longrightarrow tps1\ !\ j = (\lfloor [] \rfloor,\ 1)$
using *tps1-def k tps0* **by** *simp-all*

lemma $tps1'$: $tps1\ !\ 0 = (\lfloor xs \rfloor,\ 1)$
proof −
have $tps1\ !\ 0 = tps0\ !\ 0\ |+|\ 1$
using *tps1-def k lentps1*

by (*smt* (*verit*, *del-insts*) *add.right-neutral length-greater-0-conv length-map less-or-eq-imp-le list.size*(*3*) *not-numeral-le-zero nth-map nth-upt zero-less-one*)
then show *?thesis*
using *tps0* **by** *simp*
qed

lemma *tm1* [*transforms-intros*]: *transforms tm1 tps0 1 tps1*
unfolding *tm1-def* **by** (*tform tps*: *tps1-def tps0 k*)

abbreviation *zs* ≡ *zeros m*

definition *tpsA* ≡ *tps1*
[*9* := *exc zs T′* <!> *0*,
3 := (⌊*exc zs T′* <#> *0*⌋$_N$, *1*),
6 := (⌊*exc zs T′* <#> *1*⌋$_N$, *1*),
10 := *exc zs T′* <!> *1*]

lemma *tpsA*:
tpsA ! *0* = (⌊*xs*⌋, *1*)
tpsA ! *1* = (⌊[]⌋, *0*)
10 < *j* ⟹ *j* < *110* ⟹ *tpsA* ! *j* = (⌊[]⌋, *1*)
using *tpsA-def tps1 tps1′* **by** *simp-all*

lemma *lentpsA*: *length tpsA* = *110*
using *tpsA-def tps1-def k* **by** *simp*

definition *tps2* ≡ *tpsA*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T′*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
18 := (⌊*m′*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m′* ∗ *H*⌋$_N$, *1*)]

lemma *lentps2*: *length tps2* = *110*
using *tps2-def lentpsA* **by** *simp*

lemma *tm2* [*transforms-intros*]:
assumes *ttt* = *1* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m′*) ^ *4*
shows *transforms tm2 tps0 ttt tps2*
unfolding *tm2-def*
proof (*tform tps*: *tps0 tps1-def k lentps1*)
have *m′*: *m′* = *T′* + *Suc* (*2* ∗ *Suc* (*n* + *p n*))
using *m′-def* **by** *simp*
show *ttt* = *1* + ((*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *T′* + *Suc* (*Suc* (*Suc* (*2* ∗ *n* + *2* ∗ *p n*)))) ^ *4*)
using *assms m′*
by (*metis ab-semigroup-add-class.add-ac*(*1*) *add-2-eq-Suc distrib-left-numeral mult-Suc-right*)
have *m*: *m* = *2* ∗ *Suc* (*n* + *p n*)
using *m-def* **by** *simp*
show *tps2* = *tps1*
[*9* := *exc zs T′* <!> *0*,
3 := (⌊*snd* (*exc zs T′*) :#: *0*⌋$_N$, *1*),
6 := (⌊*snd* (*exc zs T′*) :#: *1*⌋$_N$, *1*),
10 := *exc zs T′* <!> *1*, *11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*2* ∗ *Suc* (*n* + *p n*)⌋$_N$, *1*),
17 := (⌊*T′*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *snd* (*exc zs t*) :#: *0*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *snd* (*exc zs t*) :#: *1*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
18 := (⌊*T′* + *Suc* (*2* ∗ *Suc* (*n* + *p n*))⌋$_N$, *1*),

$19 := (\lfloor H\rfloor_N,\ 1)$,
$20 := (\lfloor (T' + Suc\ (2 * Suc\ (n + p\ n))) * H\rfloor_N,\ 1)]$
using *tps2-def tpsA-def m m'* **by** *presburger*
qed

definition *tps3* $\equiv$ *tpsA*
$[11 := (\lfloor n\rfloor_N,\ 1)$,
$15 := (\lfloor p\ n\rfloor_N,\ 1)$,
$16 := (\lfloor m\rfloor_N,\ 1)$,
$17 := (\lfloor T'\rfloor_N,\ 1)$,
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T']\rfloor_{NL},\ 1)$,
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T']\rfloor_{NL},\ 1)$,
$18 := (\lfloor m'\rfloor_N,\ 1)$,
$19 := (\lfloor H\rfloor_N,\ 1)$,
$20 := (\lfloor m' * H\rfloor_N,\ 1)$,
$21 := (\lfloor m'\rfloor_N,\ 1)]$

lemma *lentps3*: *length tps3* = *110*
using *tps3-def lentpsA* **by** *simp*

lemma *tm3* [*transforms-intros*]:
assumes $ttt = 15 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m'$
shows *transforms tm3 tps0 ttt tps3*
unfolding *tm3-def*
proof (*tform tps*: *lentps2 k assms tps2-def*)
have *tps2* ! *21* = *tpsA* ! *21*
using *tps2-def* **by** *simp*
then show $tps2\ !\ 21 = (\lfloor 0\rfloor_N,\ 1)$
using *tpsA canrepr-0 k lentps1* **by** *simp*
show $tps3 = tps2[21 := (\lfloor m'\rfloor_N,\ 1)]$
unfolding *tps3-def tps2-def* **by** (*simp only*:)
show $ttt = 1 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + (14 + 3 * (nlength\ m' + nlength\ 0))$
using *assms* **by** *simp*
qed

definition *tps4* $\equiv$ *tpsA*
$[11 := (\lfloor n\rfloor_N,\ 1)$,
$15 := (\lfloor p\ n\rfloor_N,\ 1)$,
$16 := (\lfloor m\rfloor_N,\ 1)$,
$17 := (\lfloor T'\rfloor_N,\ 1)$,
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T']\rfloor_{NL},\ 1)$,
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T']\rfloor_{NL},\ 1)$,
$18 := (\lfloor m'\rfloor_N,\ 1)$,
$19 := (\lfloor H\rfloor_N,\ 1)$,
$20 := (\lfloor m' * H\rfloor_N,\ 1)$,
$21 := (\lfloor m'\rfloor_N,\ 1)$,
$22 := (\lfloor H\rfloor_N,\ 1)]$

lemma *lentps4*: *length tps4* = *110*
using *tps4-def lentpsA* **by** (*simp only*: *length-list-update*)

lemma *tm4* [*transforms-intros*]:
assumes $ttt = 29 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 3 * nlength\ H$
shows *transforms tm4 tps0 ttt tps4*
unfolding *tm4-def*
proof (*tform*)
show $19 < length\ tps3\ 22 < length\ tps3$
using *lentps3 k* **by** *simp-all*
show $tps3\ !\ 19 = (\lfloor H\rfloor_N,\ 1)$
using *tps3-def lentps3 k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
have *tps3* ! *22* = *tpsA* ! *22*
unfolding *tps3-def* **by** *simp*
then show $tps3\ !\ 22 = (\lfloor 0\rfloor_N,\ 1)$

using *tpsA canrepr-0 k lentps1* **by** *simp*
show *tps4* = *tps3*[*22* := (⌊*H*⌋$_N$, *1*)]
unfolding *tps4-def tps3-def* **by** (*simp only*:)
show *ttt* = *15* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m′*) ^ *4* +
3 ∗ *nlength m′* + (*14* + *3* ∗ (*nlength H* + *nlength 0*))
using *assms* **by** *simp*
qed

definition *tps5* ≡ *tpsA*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T′*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
18 := (⌊*m′*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m′* ∗ *H*⌋$_N$, *1*),
21 := (⌊*m′*⌋$_N$, *1*),
22 := (⌊*H*⌋$_N$, *1*),
1 := (⌊[]⌋, *1*)]

lemma *lentps5*: *length tps5* = *110*
using *tps5-def lentpsA* **by** (*simp only*: *length-list-update*)

lemma *tm5* [*transforms-intros*]:
assumes *ttt* = *30* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m′*) ^ *4* + *3* ∗ *nlength m′* + *3* ∗ *nlength H*
shows *transforms tm5 tps0 ttt tps5*
unfolding *tm5-def*
proof (*tform*)
show *1* < *length tps4*
using *lentps4 k* **by** *simp*
show *ttt* = *29* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m′*) ^ *4* + *3* ∗ *nlength m′* + *3* ∗ *nlength H* + *Suc 0*
using *assms* **by** *simp*
have *tps4* ! *1* = *tpsA* ! *1*
using *tps4-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then have *tps4* ! *1* = (⌊[]⌋, *0*)
using *tpsA* **by** *simp*
then have *tps4* ! *1* |+| *1* = (⌊[]⌋, *1*)
by *simp*
then show *tps5* = *tps4*[*1* := *tps4* ! *1* |+| *1*]
unfolding *tps5-def tps4-def* **by** (*simp only*: *list-update-swap*)
qed

definition *tps6* ≡ *tpsA*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T′*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
18 := (⌊*m′*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m′* ∗ *H*⌋$_N$, *1*),
21 := (⌊*m′*⌋$_N$, *1*),
22 := (⌊*H*⌋$_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0),
21 := (⌊*Suc* (*Suc m′*)⌋$_N$, *1*),
21 + *2* := (⌊*0*⌋$_N$, *1*),
21 + *6* := (⌊*nll-Psi* (*Suc* (*Suc m′*) ∗ *H*) *H 0*⌋$_{NLL}$, *1*)]

lemma *lentps6*: *length tps6* = *110*
using *tps6-def lentpsA* **by** (*simp only*: *length-list-update*)

lemma *tm6* [*transforms-intros*]:
assumes *ttt* = *30* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *3* ∗ *nlength H* + *5627* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2
shows *transforms tm6 tps0 ttt tps6*
unfolding *tm6-def*
proof (*tform*)
show *21* + *8* < *length tps5*
using *k lentps5* **by** *simp*
show *tps5* ! *1* = (⌊[]⌋, *1*)
using *tps5-def lentps5 k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
show *tps5* ! *21* = ($\lfloor m' \rfloor_N$, *1*)
using *tps5-def lentps5 k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
have ∗: *21* + *1* = *22*
by *simp*
show *tps5* ! (*21* + *1*) = ($\lfloor H \rfloor_N$, *1*)
using *tps5-def lentps5 k* **by** (*simp only*: ∗ *length-list-update nth-list-update-eq nth-list-update-neq*)
have *tps5* ! *23* = *tpsA* ! *23*
using *tps5-def* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq*)
then show *tps5* ! (*21* + *2*) = (⌊[]⌋, *1*)
using *tpsA k* **by** *simp*
have *tps5* ! *24* = *tpsA* ! *24*
using *tps5-def* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq*)
then show *tps5* ! (*21* + *3*) = (⌊[]⌋, *1*)
using *tpsA k* **by** *simp*
have *tps5* ! *25* = *tpsA* ! *25*
using *tps5-def* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq*)
then show *tps5* ! (*21* + *4*) = (⌊[]⌋, *1*)
using *tpsA k* **by** *simp*
have *tps5* ! *26* = *tpsA* ! *26*
using *tps5-def* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq*)
then show *tps5* ! (*21* + *5*) = (⌊[]⌋, *1*)
using *tpsA k* **by** *simp*
have *tps5* ! *27* = *tpsA* ! *27*
using *tps5-def* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq*)
then show *tps5* ! (*21* + *6*) = (⌊[]⌋, *1*)
using *tpsA k* **by** *simp*
have *tps5* ! *28* = *tpsA* ! *28*
using *tps5-def* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq*)
then show *tps5* ! (*21* + *7*) = (⌊[]⌋, *1*)
using *tpsA k* **by** *simp*
have *tps5* ! *29* = *tpsA* ! *29*
using *tps5-def* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq*)
then show *tps5* ! (*21* + *8*) = (⌊[]⌋, *1*)
using *tpsA k* **by** *simp*
show *ttt* = *30* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* +
3 ∗ *nlength m'* + *3* ∗ *nlength H* + *5627* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2
using *assms* **by** *simp*
show *tps6* = *tps5*
[*21* := ($\lfloor Suc\ (Suc\ m') \rfloor_N$, *1*),
21 + *2* := ($\lfloor 0 \rfloor_N$, *1*),
21 + *6* := ($\lfloor nll\text{-}Psi\ (Suc\ (Suc\ m') * H)\ H\ 0 \rfloor_{NLL}$, *1*),
1 := *nlltape* (*formula-n* Φ_0)]
unfolding *tps6-def tps5-def* **by** (*simp only*: *list-update-swap*[*of 1*] *list-update-overwrite*)
qed

definition *tpsB* ≡ *tpsA*
[*21* := ($\lfloor m' \rfloor_N$, *1*),
22 := ($\lfloor H \rfloor_N$, *1*),
21 := ($\lfloor Suc\ (Suc\ m') \rfloor_N$, *1*),
21 + *2* := ($\lfloor 0 \rfloor_N$, *1*),
21 + *6* := ($\lfloor nll\text{-}Psi\ (Suc\ (Suc\ m') * H)\ H\ 0 \rfloor_{NLL}$, *1*)]

lemma *tpsB*: $j > 27 \Longrightarrow j < 110 \Longrightarrow tpsB\ !\ j = (\lfloor[]\rfloor,\ 1)$
  using *tpsB-def tpsA* **by** *simp*

lemma *lentpsB*: $length\ tpsB = 110$
  using *lentpsA tpsB-def* **by** *simp*

lemma *tps6*: $tps6 = tpsB$
  $[11 := (\lfloor n\rfloor_N,\ 1),$
   $15 := (\lfloor p\ n\rfloor_N,\ 1),$
   $16 := (\lfloor m\rfloor_N,\ 1),$
   $17 := (\lfloor T'\rfloor_N,\ 1),$
   $4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T']\rfloor_{NL},\ 1),$
   $7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T']\rfloor_{NL},\ 1),$
   $18 := (\lfloor m'\rfloor_N,\ 1),$
   $19 := (\lfloor H\rfloor_N,\ 1),$
   $20 := (\lfloor m' * H\rfloor_N,\ 1),$
   $1 := nlltape\ (formula\text{-}n\ \Phi_0)]$
  unfolding *tps6-def tpsB-def* **by** (*simp only*: *list-update-swap*)

lemma *tps6'*: $j > 27 \Longrightarrow j < 110 \Longrightarrow tps6\ !\ j = (\lfloor[]\rfloor,\ 1)$
  using *tps6 tpsB* **by** (*simp only*: *nth-list-update-neq*)

definition $tps7 \equiv tpsB$
  $[11 := (\lfloor n\rfloor_N,\ 1),$
   $15 := (\lfloor p\ n\rfloor_N,\ 1),$
   $16 := (\lfloor m\rfloor_N,\ 1),$
   $17 := (\lfloor T'\rfloor_N,\ 1),$
   $4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T']\rfloor_{NL},\ 1),$
   $7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T']\rfloor_{NL},\ 1),$
   $18 := (\lfloor m'\rfloor_N,\ 1),$
   $19 := (\lfloor H\rfloor_N,\ 1),$
   $20 := (\lfloor m' * H\rfloor_N,\ 1),$
   $1 := nlltape\ (formula\text{-}n\ \Phi_0),$
   $29 := (\lfloor H\rfloor_N,\ 1)]$

lemma *tm7* [*transforms-intros*]:
  assumes $ttt = 40 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 3 * nlength\ H +$
    $5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 2 * nlength\ H$
  shows *transforms tm7 tps0 ttt tps7*
  unfolding *tm7-def*
proof (*tform*)
  show $29 < length\ tps6$
    using *lentpsB k tps6* **by** (*simp only*: *length-list-update*)
  show $tps6\ !\ 29 = (\lfloor 0\rfloor_N,\ 1)$
    using *tps6' canrepr-0 k* **by** *simp*
  show $tps7 = tps6[29 := (\lfloor H\rfloor_N,\ 1)]$
    unfolding *tps7-def* **using** *tps6* **by** (*simp only*: *list-update-swap*)
  show $ttt = 30 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 +$
      $3 * nlength\ m' + 3 * nlength\ H + 5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 +$
      $(10 + 2 * nlength\ 0 + 2 * nlength\ H)$
    using *assms* **by** *simp*
qed

definition $tps8 \equiv tpsB$
  $[11 := (\lfloor n\rfloor_N,\ 1),$
   $15 := (\lfloor p\ n\rfloor_N,\ 1),$
   $16 := (\lfloor m\rfloor_N,\ 1),$
   $17 := (\lfloor T'\rfloor_N,\ 1),$
   $4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T']\rfloor_{NL},\ 1),$
   $7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T']\rfloor_{NL},\ 1),$
   $18 := (\lfloor m'\rfloor_N,\ 1),$
   $19 := (\lfloor H\rfloor_N,\ 1),$
   $20 := (\lfloor m' * H\rfloor_N,\ 1),$

$1 := nlltape\ (formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1),$
$29 := (\lfloor H\rfloor_N,\ 1),$
$28 + 2 := (\lfloor 1\rfloor_N,\ 1),$
$28 + 6 := (\lfloor nll\text{-}Psi\ 0\ H\ 1\rfloor_{NLL},\ 1)]$

lemma *tm8* [*transforms-intros*]:
assumes $ttt = 40 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 +$
$3 * nlength\ m' + 3 * nlength\ H + 5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 +$
$2 * nlength\ H + 1875 * H\ \hat{}\ 4$
shows *transforms tm8 tps0 ttt tps8*
unfolding *tm8-def*
proof (*tform*)
show $28 + 7 <$ *length tps7*
using *lentpsB k tps7-def* **by** (*simp only*: *length-list-update*)
show *tps7* ! $1 =$ *nlltape* (*formula-n* Φ_0)
using *tps7-def lentpsB k* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq length-list-update*)
show *tps7* ! $28 = (\lfloor 0\rfloor_N,\ 1)$
using *tpsB tps7-def canrepr-0 k* **by** (*simp only*: *nth-list-update-neq*)
have *tps7* ! $29 = (\lfloor H\rfloor_N,\ 1)$
using *tps7-def lentpsB k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps7* ! $(28 + 1) = (\lfloor H\rfloor_N,\ 1)$
by *simp*
have *tps7* ! $30 =$ *tpsB* ! 30
using *tps7-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps7* ! $(28 + 2) = (\lfloor[]\rfloor,\ 1)$
using *tpsB k* **by** *simp*
have *tps7* ! $31 =$ *tpsB* ! 31
using *tps7-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps7* ! $(28 + 3) = (\lfloor[]\rfloor,\ 1)$
using *tpsB k* **by** *simp*
have *tps7* ! $32 =$ *tpsB* ! 32
using *tps7-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps7* ! $(28 + 4) = (\lfloor[]\rfloor,\ 1)$
using *tpsB k* **by** *simp*
have *tps7* ! $33 =$ *tpsB* ! 33
using *tps7-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps7* ! $(28 + 5) = (\lfloor[]\rfloor,\ 1)$
using *tpsB k* **by** *simp*
have *tps7* ! $34 =$ *tpsB* ! 34
using *tps7-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps7* ! $(28 + 6) = (\lfloor[]\rfloor,\ 1)$
using *tpsB k* **by** *simp*
have *tps7* ! $35 =$ *tpsB* ! 35
using *tps7-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps7* ! $(28 + 7) = (\lfloor[]\rfloor,\ 1)$
using *tpsB k* **by** *simp*
show $ttt = 40 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 +$
$3 * nlength\ m' + 3 * nlength\ H + 5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 +$
$2 * nlength\ H + 1875 * H\ \hat{}\ 4$
using *assms* **by** *simp*
show $tps8 = tps7$
$[28 + 2 := (\lfloor 1\rfloor_N,\ 1),$
$28 + 6 := (\lfloor nll\text{-}Psi\ 0\ H\ 1\rfloor_{NLL},\ 1),$
$1 := nlltape\ (formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1)]$
unfolding *tps8-def tps7-def* **by** (*simp only*: *list-update-swap*[*of 1*] *list-update-overwrite*)
qed

definition $tpsC \equiv tpsB$
$[29 := (\lfloor H\rfloor_N,\ 1),$
$28 + 2 := (\lfloor 1\rfloor_N,\ 1),$
$28 + 6 := (\lfloor nll\text{-}Psi\ 0\ H\ 1\rfloor_{NLL},\ 1)]$

lemma *tpsC*: $j > 34 \Longrightarrow j < 110 \Longrightarrow tpsC\ !\ j = (\lfloor[]\rfloor,\ 1)$

```
  using tpsC-def tpsB by simp

lemma lentpsC: length tpsC = 110
  using lentpsB tpsC-def by simp

lemma tps8: tps8 = tpsC
  [11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊m⌋_N, 1),
   17 := (⌊T'⌋_N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
   18 := (⌊m'⌋_N, 1),
   19 := (⌊H⌋_N, 1),
   20 := (⌊m' * H⌋_N, 1),
   1 := nlltape (formula-n Φ_0 @ formula-n Φ_1)]
  unfolding tps8-def tpsC-def by (simp only: list-update-swap)

definition tps9 ≡ tpsC
  [11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊m⌋_N, 1),
   17 := (⌊T'⌋_N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
   18 := (⌊m'⌋_N, 1),
   19 := (⌊H⌋_N, 1),
   20 := (⌊m' * H⌋_N, 1),
   1 := nlltape (formula-n Φ_0 @ formula-n Φ_1),
   35 := (⌊n⌋_N, 1)]

lemma tm9 [transforms-intros]:
  assumes ttt = 54 + (2 * d-polynomial p + 826) * (H + m') ^ 4 + 3 * nlength m' +
    5 * nlength H + 5627 * H ^ 4 * (3 + nlength (3 * H + m' * H))^2 + 1875 * H ^ 4 +
    3 * nlength n
  shows transforms tm9 tps0 ttt tps9
  unfolding tm9-def
proof (tform)
  show 11 < length tps8 35 < length tps8
    using lentpsC tps8 k by (simp-all only: length-list-update)
  show tps8 ! 11 = (⌊n⌋_N, 1)
    using tps8 lentpsC k by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  have tps8 ! 35 = tpsC ! 35
    using tps8 by (simp only: nth-list-update-neq)
  then show tps8 ! 35 = (⌊0⌋_N, 1)
    using tpsC k canrepr-0 by simp
  show tps9 = tps8[35 := (⌊n⌋_N, 1)]
    unfolding tps9-def tps8 by (simp only:)
  show ttt = 40 + (2 * d-polynomial p + 826) * (H + m') ^ 4 +
      3 * nlength m' + 3 * nlength H + 5627 * H ^ 4 * (3 + nlength (3 * H + m' * H))^2 +
      2 * nlength H + 1875 * H ^ 4 + (14 + 3 * (nlength n + nlength 0))
    using assms by simp
qed

definition tps10 ≡ tpsC
  [11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊m⌋_N, 1),
   17 := (⌊T'⌋_N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
   18 := (⌊m'⌋_N, 1),
   19 := (⌊H⌋_N, 1),
```

20 := (⌊m' ∗ *H*⌋$_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1),
35 := (⌊*n*⌋$_N$, *1*),
36 := (⌊*H*⌋$_N$, *1*)]

lemma *tm10* [*transforms-intros*]:
assumes *ttt* = *64* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + m') ^ *4* +
3 ∗ *nlength* m' + *5* ∗ *nlength H* + *5627* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + m' ∗ *H*))^2 +
1875 ∗ *H* ^ *4* + *3* ∗ *nlength n* + *2* ∗ *nlength H*
shows *transforms tm10 tps0 ttt tps10*
unfolding *tm10-def*
proof (*tform*)
show *36* < *length tps9*
using *lentpsC tps9-def k* **by** (*simp-all only*: *length-list-update*)
have *tps9* ! *36* = *tpsC* ! *36*
using *tps9-def* **by** (*simp only*: *nth-list-update-neq*)
then show *tps9* ! *36* = (⌊*0*⌋$_N$, *1*)
using *tpsC k canrepr-0* **by** *simp*
show *tps10* = *tps9*[*36* := (⌊*H*⌋$_N$, *1*)]
unfolding *tps10-def tps9-def* **by** (*simp only*: *list-update-swap*)
show *ttt* = *54* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + m') ^ *4* +
3 ∗ *nlength* m' + *5* ∗ *nlength H* + *5627* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + m' ∗ *H*))^2 +
1875 ∗ *H* ^ *4* + *3* ∗ *nlength n* + (*10* + *2* ∗ *nlength 0* + *2* ∗ *nlength H*)
using *assms* **by** *simp*
qed

definition *tps11* ≡ *tpsC*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊T'⌋$_N$, *1*),
4 := (⌊*map* (λ*t. exc zs t* <#> *0*) [*0..*<*Suc* T']⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t. exc zs t* <#> *1*) [*0..*<*Suc* T']⌋$_{NL}$, *1*),
18 := (⌊m'⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊m' ∗ *H*⌋$_N$, *1*),
1 := *nlltape* ((*formula-n* Φ_0 @ *formula-n* Φ_1) @ *formula-n* Φ_2),
35 := (⌊*n*⌋$_N$, *1*),
36 := (⌊*H*⌋$_N$, *1*),
35 := (⌊*2* ∗ *n* + *2*⌋$_N$, *1*),
35 + *2* := (⌊*3*⌋$_N$, *1*),
35 + *6* := (⌊*nll-Psi* (*Suc* (*Suc* (*2* ∗ *n*)) ∗ *H*) *H 3*⌋$_{NLL}$, *1*)]

lemma *tm11* [*transforms-intros*]:
assumes *ttt* = *64* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + m') ^ *4* +
3 ∗ *nlength* m' + *7* ∗ *nlength H* + *5627* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + m' ∗ *H*))^2 +
1875 ∗ *H* ^ *4* + *3* ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2
shows *transforms tm11 tps0 ttt tps11*
unfolding *tm11-def*
proof (*tform*)
show *35* + *8* < *length tps10*
using *lentpsC k tps10-def* **by** (*simp only*: *length-list-update*)
show *tps10* ! *1* = *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1)
using *tps10-def lentpsC k* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq length-list-update*)
show *tps10* ! *35* = (⌊*n*⌋$_N$, *1*)
using *tps10-def lentpsC k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
have *tps10* ! *36* = (⌊*H*⌋$_N$, *1*)
using *tps10-def lentpsC k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps10* ! (*35* + *1*) = (⌊*H*⌋$_N$, *1*)
by *simp*
have *tps10* ! *37* = *tpsC* ! *37*
using *tps10-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps10* ! (*35* + *2*) = (⌊[]⌋, *1*)

using *tpsC k* **by** *simp*
have *tps10* ! *38* = *tpsC* ! *38*
using *tps10-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps10* ! (*35* + *3*) = (⌊[]⌋, *1*)
using *tpsC k* **by** *simp*
have *tps10* ! *39* = *tpsC* ! *39*
using *tps10-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps10* ! (*35* + *4*) = (⌊[]⌋, *1*)
using *tpsC k* **by** *simp*
have *tps10* ! *40* = *tpsC* ! *40*
using *tps10-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps10* ! (*35* + *5*) = (⌊[]⌋, *1*)
using *tpsC k* **by** *simp*
have *tps10* ! *41* = *tpsC* ! *41*
using *tps10-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps10* ! (*35* + *6*) = (⌊[]⌋, *1*)
using *tpsC k* **by** *simp*
have *tps10* ! *42* = *tpsC* ! *42*
using *tps10-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps10* ! (*35* + *7*) = (⌊[]⌋, *1*)
using *tpsC k* **by** *simp*
have *tps10* ! *43* = *tpsC* ! *43*
using *tps10-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps10* ! (*35* + *8*) = (⌊[]⌋, *1*)
using *tpsC k* **by** *simp*
show *tps11* = *tps10*
[*35* := (⌊*2* ∗ *n* + *2*⌋$_N$, *1*),
35 + *2* := (⌊*3*⌋$_N$, *1*),
35 + *6* := (⌊*nll-Psi* (*Suc* (*Suc* (*2* ∗ *n*)) ∗ *H*) *H 3*⌋$_{NLL}$, *1*),
1 := *nlltape* ((*formula-n* Φ_0 @ *formula-n* Φ_1) @ *formula-n* Φ_2)]
unfolding *tps11-def tps10-def* **by** (*simp only*: *list-update-swap*[*of 1*] *list-update-overwrite*)
show *ttt* = *64* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* +
5 ∗ *nlength H* + *5627* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
3 ∗ *nlength n* + *2* ∗ *nlength H* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2
using *assms* **by** *simp*
qed

definition *tpsD* ≡ *tpsC*
[*35* := (⌊*n*⌋$_N$, *1*),
36 := (⌊*H*⌋$_N$, *1*),
35 := (⌊*2* ∗ *n* + *2*⌋$_N$, *1*),
35 + *2* := (⌊*3*⌋$_N$, *1*),
35 + *6* := (⌊*nll-Psi* (*Suc* (*Suc* (*2* ∗ *n*)) ∗ *H*) *H 3*⌋$_{NLL}$, *1*)]

lemma *tpsD*: *41* < *j* ⟹ *j* < *110* ⟹ *tpsD* ! *j* = (⌊[]⌋, *1*)
using *tpsD-def tpsC* **by** *simp*

lemma *lentpsD*: *length tpsD* = *110*
using *lentpsC tpsD-def* **by** *simp*

lemma *tps11*: *tps11* = *tpsD*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape* ((*formula-n* Φ_0 @ *formula-n* Φ_1) @ *formula-n* Φ_2)]
unfolding *tps11-def tpsD-def* **by** (*simp only*: *list-update-swap*)

definition *tps12* ≡ *tpsD*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape* ((*formula-n* Φ_0 @ *formula-n* Φ_1) @ *formula-n* Φ_2),
42 := (⌊*1*⌋$_N$, *1*)]

lemma *tm12* [*transforms-intros*]:
assumes *ttt* = *76* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *7* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
3 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2
shows *transforms tm12 tps0 ttt tps12*
unfolding *tm12-def*
proof (*tform*)
show *42* < *length tps11*
using *lentpsD tps11 k* **by** (*simp-all only*: *length-list-update*)
have *tps11* ! *42* = *tpsD* ! *42*
using *tps11* **by** (*simp only*: *nth-list-update-neq*)
then show *tps11* ! *42* = (⌊*0*⌋$_N$, *1*)
using *tpsD k canrepr-0* **by** *simp*
show *tps12* = *tps11*[*42* := (⌊*1*⌋$_N$, *1*)]
unfolding *tps12-def tps11* **by** (*simp only*:)
show *ttt* = *64* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *7* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
3 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
(*10* + *2* ∗ *nlength 0* + *2* ∗ *nlength 1*)
using *canrepr-1 assms* **by** *simp*
qed

definition *tps13* ≡ *tpsD*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape* ((*formula-n* Φ_0 @ *formula-n* Φ_1) @ *formula-n* Φ_2),
42 := (⌊*1*⌋$_N$, *1*),
43 := (⌊*H*⌋$_N$, *1*)]

lemma *tm13* [*transforms-intros*]:
assumes *ttt* = *86* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *9* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
3 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2
shows *transforms tm13 tps0 ttt tps13*
unfolding *tm13-def*
proof (*tform*)
show *43* < *length tps12*
using *lentpsD tps12-def k* **by** (*simp-all only*: *length-list-update*)
have *tps12* ! *43* = *tpsD* ! *43*
using *tps12-def* **by** (*simp only*: *nth-list-update-neq*)
then show *tps12* ! *43* = (⌊*0*⌋$_N$, *1*)
using *tpsD k canrepr-0* **by** *simp*
show *tps13* = *tps12*[*43* := (⌊*H*⌋$_N$, *1*)]

unfolding *tps13-def tps12-def* **by** (*simp only*:)
show $ttt = 76 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 7 * nlength\ H + 5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H\ \hat{}\ 4 + 3 * nlength\ n + 3764 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 + (10 + 2 * nlength\ 0 + 2 * nlength\ H)$
using *assms* **by** *simp*
qed

definition $tps14 \equiv tpsD$
$[11 := (\lfloor n\rfloor_N, 1),$
$15 := (\lfloor p\ n\rfloor_N, 1),$
$16 := (\lfloor m\rfloor_N, 1),$
$17 := (\lfloor T'\rfloor_N, 1),$
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T']\rfloor_{NL}, 1),$
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T']\rfloor_{NL}, 1),$
$18 := (\lfloor m'\rfloor_N, 1),$
$19 := (\lfloor H\rfloor_N, 1),$
$20 := (\lfloor m' * H\rfloor_N, 1),$
$1 := nlltape\ ((formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1)\ @\ formula\text{-}n\ \Phi_2),$
$42 := (\lfloor 1\rfloor_N, 1),$
$43 := (\lfloor H\rfloor_N, 1),$
$44 := (\lfloor 2\rfloor_N, 1)]$

lemma *tm14* [*transforms-intros*]:
assumes $ttt = 100 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 9 * nlength\ H + 5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H\ \hat{}\ 4 + 3 * nlength\ n + 3764 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + 2 * n * H))^2$
shows *transforms tm14 tps0 ttt tps14*
unfolding *tm14-def*
proof (*tform*)
show $44 < length\ tps13$
using *lentpsD tps13-def k* **by** (*simp-all only*: *length-list-update*)
have $tps13\ !\ 44 = tpsD\ !\ 44$
using *tps13-def* **by** (*simp only*: *nth-list-update-neq*)
then show $tps13\ !\ 44 = (\lfloor 0\rfloor_N, 1)$
using *tpsD k canrepr-0* **by** *simp*
show $tps14 = tps13[44 := (\lfloor 2\rfloor_N, 1)]$
unfolding *tps14-def tps13-def* **by** (*simp only*:)
show $ttt = 86 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 9 * nlength\ H + 5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H\ \hat{}\ 4 + 3 * nlength\ n + 3764 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 + (10 + 2 * nlength\ 0 + 2 * nlength\ 2)$
using *nlength-2 assms* **by** *simp*
qed

definition $tps15 \equiv tpsD$
$[11 := (\lfloor n\rfloor_N, 1),$
$15 := (\lfloor p\ n\rfloor_N, 1),$
$16 := (\lfloor m\rfloor_N, 1),$
$17 := (\lfloor T'\rfloor_N, 1),$
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T']\rfloor_{NL}, 1),$
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T']\rfloor_{NL}, 1),$
$18 := (\lfloor m'\rfloor_N, 1),$
$19 := (\lfloor H\rfloor_N, 1),$
$20 := (\lfloor m' * H\rfloor_N, 1),$
$1 := nlltape\ ((formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1)\ @\ formula\text{-}n\ \Phi_2),$
$42 := (\lfloor 1\rfloor_N, 1),$
$43 := (\lfloor H\rfloor_N, 1),$
$44 := (\lfloor 2\rfloor_N, 1),$
$50 := (\lfloor n\rfloor_N, 1)]$

lemma *tm15* [*transforms-intros*]:
assumes $ttt = 114 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 9 * nlength\ H +$

5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
6 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2
shows *transforms tm15 tps0 ttt tps15*
unfolding *tm15-def*
proof (*tform*)
show *11* < *length tps14 50* < *length tps14*
using *lentpsD tps14-def k* **by** (*simp-all only*: *length-list-update*)
show *tps14* ! *11* = (⌊*n*⌋$_N$, *1*)
using *tps14-def k lentpsD* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
have *tps14* ! *50* = *tpsD* ! *50*
using *tps14-def* **by** (*simp only*: *nth-list-update-neq*)
then show *tps14* ! *50* = (⌊*0*⌋$_N$, *1*)
using *tpsD k canrepr-0* **by** *simp*
show *tps15* = *tps14*[*50* := (⌊*n*⌋$_N$, *1*)]
unfolding *tps15-def tps14-def* **by** (*simp only*:)
show *ttt* = *100* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *9* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
3 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
(*14* + *3* ∗ (*nlength n* + *nlength 0*))
using *assms* **by** *simp*
qed

definition *tps16* ≡ *tpsD*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape* ((*formula-n* Φ_0 @ *formula-n* Φ_1) @ *formula-n* Φ_2),
42 := (⌊*1*⌋$_N$, *1*),
43 := (⌊*H*⌋$_N$, *1*),
44 := (⌊*2*⌋$_N$, *1*),
50 := (⌊*1* + *2* ∗ *n*⌋$_N$, *1*)]

lemma *tm16* [*transforms-intros*]:
assumes *ttt* = *126* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *9* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
10 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2
shows *transforms tm16 tps0 ttt tps16*
unfolding *tm16-def*
proof (*tform*)
show *2* ≤ *length tps15 50* < *length tps15*
using *lentpsD tps15-def k* **by** (*simp-all only*: *length-list-update*)
show *tps15* ! *50* = (⌊*n*⌋$_N$, *1*)
using *tps15-def k lentpsD* **by** (*simp only*: *length-list-update nth-list-update-eq*)
have *tps16* = *tps15*[*50* := (⌊*1* + *2* ∗ *n*⌋$_N$, *1*)]
unfolding *tps16-def tps15-def* **by** (*simp only*: *list-update-swap*[*of 1*] *list-update-overwrite*)
then show *tps16* = *tps15*[*50* := (⌊*Suc* (*2* ∗ *n*)⌋$_N$, *1*)]
by *simp*
show *ttt* = *114* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *9* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* + *6* ∗ *nlength n* +
3764 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 + (*12* + *4* ∗ *nlength n*)
using *assms* **by** *simp*
qed

definition *tps17* ≡ *tpsD*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),

17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3),
42 := (⌊*1*⌋$_N$, *1*),
43 := (⌊*H*⌋$_N$, *1*),
44 := (⌊*2*⌋$_N$, *1*),
50 := (⌊*1* + *2* ∗ *n*⌋$_N$, *1*),
42 := (⌊*1* + *2* ∗ *n*⌋$_N$, *1*),
42 + *3* := (⌊*1*⌋$_N$, *1*)]

lemma *tm17* [*transforms-intros*]:
assumes *ttt* = *126* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *9* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
10 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2))
shows *transforms tm17 tps0 ttt tps17*
unfolding *tm17-def*
proof (*tform transforms-intros*: *tm-PHI3*)
show *42* + *8* < *length tps16*
using *lentpsD k tps16-def* **by** (*simp only*: *length-list-update*)
show *tps16* ! *1* = *nlltape* ((*formula-n* Φ_0 @ *formula-n* Φ_1) @ *formula-n* Φ_2)
using *tps16-def lentpsD k* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq length-list-update*)
show *tps16* ! *42* = (⌊*1*⌋$_N$, *1*)
using *tps16-def lentpsD k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
have *tps16* ! *43* = (⌊*H*⌋$_N$, *1*)
using *tps16-def lentpsD k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps16* ! (*42* + *1*) = (⌊*H*⌋$_N$, *1*)
by *simp*
have *tps16* ! *44* = (⌊*2*⌋$_N$, *1*)
using *tps16-def lentpsD k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps16* ! (*42* + *2*) = (⌊*2*⌋$_N$, *1*)
by *simp*
have *tps16* ! *45* = *tpsD* ! *45*
using *tps16-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps16* ! (*42* + *3*) = (⌊[]⌋, *1*)
using *tpsD k* **by** *simp*
have *tps16* ! *46* = *tpsD* ! *46*
using *tps16-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps16* ! (*42* + *4*) = (⌊[]⌋, *1*)
using *tpsD k* **by** *simp*
have *tps16* ! *47* = *tpsD* ! *47*
using *tps16-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps16* ! (*42* + *5*) = (⌊[]⌋, *1*)
using *tpsD k* **by** *simp*
have *tps16* ! *48* = *tpsD* ! *48*
using *tps16-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps16* ! (*42* + *6*) = (⌊[]⌋, *1*)
using *tpsD k* **by** *simp*
have *tps16* ! *49* = *tpsD* ! *49*
using *tps16-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps16* ! (*42* + *7*) = (⌊[]⌋, *1*)
using *tpsD k* **by** *simp*
have *tps16* ! *50* = (⌊*1* + *2* ∗ *n*⌋$_N$, *1*)
using *tps16-def lentpsD k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps16* ! (*42* + *8*) = (⌊*1* + *2* ∗ *n*⌋$_N$, *1*)
by *simp*
have *tps17* = *tps16*
[*42* := (⌊*1* + *2* ∗ *n*⌋$_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3),

$42 + 3 := (\lfloor 1 \rfloor_N, 1)]$
unfolding *tps17-def tps16-def* **by** (*simp only*: *list-update-swap*[*of 1*] *list-update-overwrite*)
then show *tps17 = tps16*
$[42 := (\lfloor 1 + 2 * n \rfloor_N, 1),$
$1 := nlltape (((formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1)\ @\ formula\text{-}n\ \Phi_2)\ @\ formula\text{-}n\ \Phi_3),$
$42 + 3 := (\lfloor 1 \rfloor_N, 1)]$
by *simp*
show $ttt = 126 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 9 * nlength\ H +$
$5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H\ \hat{}\ 4 +$
$10 * nlength\ n + 3764 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 +$
$Suc\ n * (9 + 1897 * (H\ \hat{}\ 4 * (nlength\ (1 + 2 * n))^2))$
using *assms* **by** *simp*
qed

definition *tpsE* ≡ *tpsD*
$[42 := (\lfloor 1 \rfloor_N, 1),$
$43 := (\lfloor H \rfloor_N, 1),$
$44 := (\lfloor 2 \rfloor_N, 1),$
$50 := (\lfloor 1 + 2 * n \rfloor_N, 1),$
$42 := (\lfloor 1 + 2 * n \rfloor_N, 1),$
$42 + 3 := (\lfloor 1 \rfloor_N, 1)]$

lemma *tpsE*: $50 < j \Longrightarrow j < 110 \Longrightarrow tpsE\ !\ j = (\lfloor [] \rfloor, 1)$
using *tpsE-def tpsD* **by** *simp*

lemma *lentpsE*: *length tpsE = 110*
using *lentpsD tpsE-def* **by** *simp*

lemma *tps17*: *tps17 = tpsE*
$[11 := (\lfloor n \rfloor_N, 1),$
$15 := (\lfloor p\ n \rfloor_N, 1),$
$16 := (\lfloor m \rfloor_N, 1),$
$17 := (\lfloor T' \rfloor_N, 1),$
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T'] \rfloor_{NL}, 1),$
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T'] \rfloor_{NL}, 1),$
$18 := (\lfloor m' \rfloor_N, 1),$
$19 := (\lfloor H \rfloor_N, 1),$
$20 := (\lfloor m' * H \rfloor_N, 1),$
$1 := nlltape\ (formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1\ @\ formula\text{-}n\ \Phi_2\ @\ formula\text{-}n\ \Phi_3)]$
unfolding *tps17-def tpsE-def* **by** (*simp only*: *list-update-swap*)

definition *tps18* ≡ *tpsE*
$[11 := (\lfloor n \rfloor_N, 1),$
$15 := (\lfloor p\ n \rfloor_N, 1),$
$16 := (\lfloor m \rfloor_N, 1),$
$17 := (\lfloor T' \rfloor_N, 1),$
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T'] \rfloor_{NL}, 1),$
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T'] \rfloor_{NL}, 1),$
$18 := (\lfloor m' \rfloor_N, 1),$
$19 := (\lfloor H \rfloor_N, 1),$
$20 := (\lfloor m' * H \rfloor_N, 1),$
$1 := nlltape\ (formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1\ @\ formula\text{-}n\ \Phi_2\ @\ formula\text{-}n\ \Phi_3),$
$52 := (\lfloor H \rfloor_N, 1)]$

lemma *tm18* [*transforms-intros*]:
assumes $ttt = 136 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 11 * nlength\ H +$
$5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H\ \hat{}\ 4 +$
$10 * nlength\ n + 3764 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 +$
$Suc\ n * (9 + 1897 * (H\ \hat{}\ 4 * (nlength\ (1 + 2 * n))^2))$
shows *transforms tm18 tps0 ttt tps18*
unfolding *tm18-def*
proof (*tform*)
show *52 < length tps17*

using *lentpsE tps17 k* **by** (*simp-all only*: *length-list-update*)
have *tps17* ! *52* = *tpsE* ! *52*
using *tps17* **by** (*simp only*: *nth-list-update-neq*)
then show *tps17* ! *52* = ($\lfloor 0 \rfloor_N$, *1*)
using *tpsE k canrepr-0* **by** *simp*
show *tps18* = *tps17*[*52* := ($\lfloor H \rfloor_N$, *1*)]
unfolding *tps18-def tps17* **by** (*simp only*:)
show *ttt* = *126* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *9* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
10 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2)) +
(*10* + *2* ∗ *nlength 0* + *2* ∗ *nlength H*)
using *assms* **by** *simp*
qed

definition *tps19* ≡ *tpsE*
[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3),
52 := ($\lfloor H \rfloor_N$, *1*),
53 := ($\lfloor 2 \rfloor_N$, *1*)]

lemma *tm19* [*transforms-intros*]:
assumes *ttt* = *150* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *11* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
10 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2))
shows *transforms tm19 tps0 ttt tps19*
unfolding *tm19-def*
proof (*tform*)
show *53* < *length tps18*
using *lentpsE tps18-def k* **by** (*simp-all only*: *length-list-update*)
have *tps18* ! *53* = *tpsE* ! *53*
using *tps18-def* **by** (*simp only*: *nth-list-update-neq*)
then show *tps18* ! *53* = ($\lfloor 0 \rfloor_N$, *1*)
using *tpsE k canrepr-0* **by** *simp*
show *tps19* = *tps18*[*53* := ($\lfloor 2 \rfloor_N$, *1*)]
unfolding *tps19-def tps18-def* **by** (*simp only*:)
show *ttt* = *136* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *11* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
10 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2)) +
(*10* + *2* ∗ *nlength 0* + *2* ∗ *nlength 2*)
using *assms nlength-2* **by** *simp*
qed

definition *tps20* ≡ *tpsE*
[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),

1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3),
52 := ($\lfloor H \rfloor_N$, *1*),
53 := ($\lfloor 2 \rfloor_N$, *1*),
51 := ($\lfloor n \rfloor_N$, *1*)]

lemma *tm20* [*transforms-intros*]:
assumes *ttt* = *164* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *11* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
13 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2))
shows *transforms tm20 tps0 ttt tps20*
unfolding *tm20-def*
proof (*tform*)
show *11* < *length tps19 51* < *length tps19*
using *lentpsE tps19-def k* **by** (*simp-all only*: *length-list-update*)
show *tps19* ! *11* = ($\lfloor n \rfloor_N$, *1*)
using *tps19-def k lentpsE* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
have *tps19* ! *51* = *tpsE* ! *51*
using *tps19-def* **by** (*simp only*: *nth-list-update-neq*)
then show *tps19* ! *51* = ($\lfloor 0 \rfloor_N$, *1*)
using *tpsE k canrepr-0* **by** *simp*
show *tps20* = *tps19*[*51* := ($\lfloor n \rfloor_N$, *1*)]
unfolding *tps20-def tps19-def* **by** (*simp only*:)
show *ttt* = *150* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *11* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
10 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2)) +
(*14* + *3* ∗ (*nlength n* + *nlength 0*))
using *assms* **by** *simp*
qed

definition *tps21* ≡ *tpsE*
[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3),
52 := ($\lfloor H \rfloor_N$, *1*),
53 := ($\lfloor 2 \rfloor_N$, *1*),
51 := ($\lfloor 2 * n \rfloor_N$, *1*)]

lemma *tm21* [*transforms-intros*]:
assumes *ttt* = *169* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *11* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
15 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2))
shows *transforms tm21 tps0 ttt tps21*
unfolding *tm21-def*
proof (*tform time*: *assms*)
show *51* < *length tps20*
using *lentpsE tps20-def k* **by** (*simp only*: *length-list-update*)
show *tps20* ! *51* = ($\lfloor n \rfloor_N$, *1*)
using *tps20-def k lentpsE* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
show *tps21* = *tps20*[*51* := ($\lfloor 2 * n \rfloor_N$, *1*)]
unfolding *tps21-def tps20-def* **by** (*simp only*: *list-update-overwrite*)
qed

definition *tps22* ≡ *tpsE*

[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3),
52 := ($\lfloor H \rfloor_N$, *1*),
53 := ($\lfloor 2 \rfloor_N$, *1*),
51 := ($\lfloor 2 * n + 3 \rfloor_N$, *1*)]

lemma *tm22* [*transforms-intros*]:
assumes *ttt* = *184* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m′*) ^ *4* + *3* ∗ *nlength m′* + *11* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m′* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
15 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2)) + *6* ∗ *nlength* (*2* ∗ *n* + *3*)
shows *transforms tm22 tps0 ttt tps22*
unfolding *tm22-def*
proof (*tform time*: *assms*)
show *51* < *length tps21*
using *lentpsE tps21-def k* **by** (*simp only*: *length-list-update*)
show *tps21* ! *51* = ($\lfloor 2 * n \rfloor_N$, *1*)
using *tps21-def k lentpsE* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
show *tps22* = *tps21*[*51* := ($\lfloor 2 * n + 3 \rfloor_N$, *1*)]
unfolding *tps22-def tps21-def* **by** (*simp only*: *list-update-overwrite*)
qed

definition *tps23* ≡ *tpsE*
[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3),
52 := ($\lfloor H \rfloor_N$, *1*),
53 := ($\lfloor 2 \rfloor_N$, *1*),
51 := ($\lfloor 2 * n + 3 \rfloor_N$, *1*),
59 := ($\lfloor m \rfloor_N$, *1*)]

lemma *tm23* [*transforms-intros*]:
assumes *ttt* = *198* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m′*) ^ *4* + *3* ∗ *nlength m′* + *11* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m′* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
15 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2)) + *6* ∗ *nlength* (*2* ∗ *n* + *3*) +
3 ∗ *nlength m*
shows *transforms tm23 tps0 ttt tps23*
unfolding *tm23-def*
proof (*tform*)
show *16* < *length tps22 59* < *length tps22*
using *lentpsE tps22-def k* **by** (*simp-all only*: *length-list-update*)
show *tps22* ! *16* = ($\lfloor m \rfloor_N$, *1*)
using *tps22-def k lentpsE* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
have *tps22* ! *59* = *tpsE* ! *59*
using *tps22-def* **by** (*simp only*: *nth-list-update-neq*)
then show *tps22* ! *59* = ($\lfloor 0 \rfloor_N$, *1*)
using *tpsE k canrepr-0* **by** *simp*

show $tps23 = tps22[59 := (\lfloor m \rfloor_N, 1)]$
unfolding *tps23-def tps22-def* **by** (*simp only*:)
show $ttt = 184 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \widehat{}\ 4 + 3 * nlength\ m' + 11 * nlength\ H + 5627 * H\ \widehat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H\ \widehat{}\ 4 + 15 * nlength\ n + 3764 * H\ \widehat{}\ 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 + Suc\ n * (9 + 1897 * (H\ \widehat{}\ 4 * (nlength\ (1 + 2 * n))^2)) + 6 * nlength\ (2 * n + 3) + (14 + 3 * (nlength\ m + nlength\ 0))$
using *assms* **by** *simp*
qed

definition $tps24 \equiv tpsE$
$[11 := (\lfloor n \rfloor_N, 1),$
$15 := (\lfloor p\ n \rfloor_N, 1),$
$16 := (\lfloor m \rfloor_N, 1),$
$17 := (\lfloor T' \rfloor_N, 1),$
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T'] \rfloor_{NL}, 1),$
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T'] \rfloor_{NL}, 1),$
$18 := (\lfloor m' \rfloor_N, 1),$
$19 := (\lfloor H \rfloor_N, 1),$
$20 := (\lfloor m' * H \rfloor_N, 1),$
$1 := nlltape\ (formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1\ @\ formula\text{-}n\ \Phi_2\ @\ formula\text{-}n\ \Phi_3),$
$52 := (\lfloor H \rfloor_N, 1),$
$53 := (\lfloor 2 \rfloor_N, 1),$
$51 := (\lfloor 2 * n + 3 \rfloor_N, 1),$
$59 := (\lfloor Suc\ m \rfloor_N, 1)]$

lemma *tm24* [*transforms-intros*]:
assumes $ttt = 203 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \widehat{}\ 4 + 3 * nlength\ m' + 11 * nlength\ H + 5627 * H\ \widehat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H\ \widehat{}\ 4 + 15 * nlength\ n + 3764 * H\ \widehat{}\ 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 + Suc\ n * (9 + 1897 * (H\ \widehat{}\ 4 * (nlength\ (1 + 2 * n))^2)) + 6 * nlength\ (2 * n + 3) + 5 * nlength\ m$
shows *transforms tm24 tps0 ttt tps24*
unfolding *tm24-def*
proof (*tform time*: *assms*)
show $59 < length\ tps23$
using *lentpsE tps23-def k* **by** (*simp only*: *length-list-update*)
have $tps23\ !\ 59 = (\lfloor m \rfloor_N, 1)$
using *tps23-def k lentpsE* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show $tps23\ !\ 59 = (\lfloor m \rfloor_N, 1)$
by *simp*
show $tps24 = tps23[59 := (\lfloor Suc\ m \rfloor_N, 1)]$
unfolding *tps24-def tps23-def* **by** (*simp only*: *list-update-overwrite*)
qed

definition $tps25 \equiv tpsE$
$[11 := (\lfloor n \rfloor_N, 1),$
$15 := (\lfloor p\ n \rfloor_N, 1),$
$16 := (\lfloor m \rfloor_N, 1),$
$17 := (\lfloor T' \rfloor_N, 1),$
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T'] \rfloor_{NL}, 1),$
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T'] \rfloor_{NL}, 1),$
$18 := (\lfloor m' \rfloor_N, 1),$
$19 := (\lfloor H \rfloor_N, 1),$
$20 := (\lfloor m' * H \rfloor_N, 1),$
$1 := nlltape\ (formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1\ @\ formula\text{-}n\ \Phi_2\ @\ formula\text{-}n\ \Phi_3\ @\ formula\text{-}n\ \Phi_4),$
$52 := (\lfloor H \rfloor_N, 1),$
$53 := (\lfloor 2 \rfloor_N, 1),$
$51 := (\lfloor 2 * n + 3 \rfloor_N, 1),$
$59 := (\lfloor Suc\ m \rfloor_N, 1),$
$51 := (\lfloor 2 * n + 2 + 1 + 2 * p\ n \rfloor_N, 1),$
$51 + 3 := (\lfloor 1 \rfloor_N, 1)]$

lemma *tm25* [*transforms-intros*]:
assumes $ttt = 203 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 11 * nlength\ H +$
$5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H\ \hat{}\ 4 +$
$15 * nlength\ n + 3764 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 +$
$Suc\ n * (9 + 1897 * (H\ \hat{}\ 4 * (nlength\ (1 + 2 * n))^2)) + 6 * nlength\ (2 * n + 3) +$
$5 * nlength\ m + Suc\ (p\ n) * (9 + 1897 * (H\ \hat{}\ 4 * (nlength\ (Suc\ m))^2))$
shows *transforms tm25 tps0 ttt tps25*
unfolding *tm25-def*
proof (*tform transforms-intros*: *tm-PHI4*)
show $51 + 8 < length\ tps24$
using *lentpsE tps24-def k* **by** (*simp only*: *length-list-update*)
show $tps24\ !\ 1 = nlltape\ (formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1\ @\ formula\text{-}n\ \Phi_2\ @\ formula\text{-}n\ \Phi_3)$
using *tps24-def lentpsE k* **by** (*simp only*: *nth-list-update-eq nth-list-update-neq length-list-update*)
have $tps24\ !\ 51 = (\lfloor 2 * n + 3 \rfloor_N, 1)$
using *tps24-def lentpsE k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show $tps24\ !\ 51 = (\lfloor 2 * n + 2 + 1 \rfloor_N, 1)$
by (*metis add.assoc nat-1-add-1 numeral-Bit1 numerals*(*1*))
have $tps24\ !\ 52 = (\lfloor H \rfloor_N, 1)$
using *tps24-def lentpsE k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show $tps24\ !\ (51 + 1) = (\lfloor H \rfloor_N, 1)$
by *simp*
have $tps24\ !\ 53 = (\lfloor 2 \rfloor_N, 1)$
using *tps24-def lentpsE k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show $tps24\ !\ (51 + 2) = (\lfloor 2 \rfloor_N, 1)$
by *simp*
have $tps24\ !\ 54 = tpsE\ !\ 54$
using *tps24-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show $tps24\ !\ (51 + 3) = (\lfloor [] \rfloor, 1)$
using *tpsE k* **by** *simp*
have $tps24\ !\ 55 = tpsE\ !\ 55$
using *tps24-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show $tps24\ !\ (51 + 4) = (\lfloor [] \rfloor, 1)$
using *tpsE k* **by** *simp*
have $tps24\ !\ 56 = tpsE\ !\ 56$
using *tps24-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show $tps24\ !\ (51 + 5) = (\lfloor [] \rfloor, 1)$
using *tpsE k* **by** *simp*
have $tps24\ !\ 57 = tpsE\ !\ 57$
using *tps24-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show $tps24\ !\ (51 + 6) = (\lfloor [] \rfloor, 1)$
using *tpsE k* **by** *simp*
have $tps24\ !\ 58 = tpsE\ !\ 58$
using *tps24-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show $tps24\ !\ (51 + 7) = (\lfloor [] \rfloor, 1)$
using *tpsE k* **by** *simp*
have $tps24\ !\ 59 = (\lfloor Suc\ m \rfloor_N, 1)$
using *tps24-def lentpsE k* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
moreover have $*$: $Suc\ m = 2 * n + 2 + 1 + 2 * p\ n$
using *m-def* **by** *simp*
ultimately show $tps24\ !\ (51 + 8) = (\lfloor 2 * n + 2 + 1 + 2 * p\ n \rfloor_N, 1)$
by *simp*
have $tps25 = tps24$
$[51 := (\lfloor 2 * n + 2 + 1 + 2 * p\ n \rfloor_N, 1),$
$1 := nlltape\ (formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1\ @\ formula\text{-}n\ \Phi_2\ @\ formula\text{-}n\ \Phi_3\ @\ formula\text{-}n\ \Phi_4),$
$51 + 3 := (\lfloor 1 \rfloor_N, 1)]$
unfolding *tps25-def tps24-def* **by** (*simp only*: *list-update-swap list-update-overwrite*)
then show $tps25 = tps24$
$[51 := (\lfloor 2 * n + 2 + 1 + 2 * p\ n \rfloor_N, 1),$
$1 := nlltape\ ((formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1\ @\ formula\text{-}n\ \Phi_2\ @\ formula\text{-}n\ \Phi_3)\ @\ formula\text{-}n\ \Phi_4),$
$51 + 3 := (\lfloor 1 \rfloor_N, 1)]$
by *simp*
show $ttt = 203 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 11 * nlength\ H +$
$5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H\ \hat{}\ 4 +$

$15 * nlength\ n + 3764 * H \hat{~} 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 +$
$Suc\ n * (9 + 1897 * (H \hat{~} 4 * (nlength\ (1 + 2 * n))^2)) + 6 * nlength\ (2 * n + 3) +$
$5 * nlength\ m + Suc\ (p\ n) * (9 + 1897 * (H \hat{~} 4 * (nlength\ (2 * n + 2 + 1 + 2 * p\ n))^2))$
using *assms* * **by** *simp*
qed

definition *tpsF* ≡ *tpsE*
[*52* := (⌊*H*⌋$_N$, *1*),
53 := (⌊*2*⌋$_N$, *1*),
51 := (⌊*2* ∗ *n* + *3*⌋$_N$, *1*),
59 := (⌊*Suc m*⌋$_N$, *1*),
51 := (⌊*2* ∗ *n* + *2* + *1* + *2* ∗ *p n*⌋$_N$, *1*),
51 + *3* := (⌊*1*⌋$_N$, *1*)]

lemma *tpsF*: *59* < *j* ⟹ *j* < *110* ⟹ *tpsF* ! *j* = (⌊[]⌋, *1*)
using *tpsF-def tpsE* **by** *simp*

lemma *lentpsF*: *length tpsF* = *110*
using *lentpsE tpsF-def* **by** *simp*

lemma *tps25*: *tps25* = *tpsF*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4)]
unfolding *tps25-def tpsF-def* **by** (*simp only*: *list-update-swap*)

definition *tps26* ≡ *tpsF*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4),
61 := (⌊*H*⌋$_N$, *1*)]

lemma *tm26* [*transforms-intros*]:
assumes $ttt = 213 + (2 * d\text{-}polynomial\ p + 826) * (H + m') \hat{~} 4 + 3 * nlength\ m' + 13 * nlength\ H +$
$5627 * H \hat{~} 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H \hat{~} 4 +$
$15 * nlength\ n + 3764 * H \hat{~} 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 +$
$Suc\ n * (9 + 1897 * (H \hat{~} 4 * (nlength\ (1 + 2 * n))^2)) + 6 * nlength\ (2 * n + 3) +$
$5 * nlength\ m + Suc\ (p\ n) * (9 + 1897 * (H \hat{~} 4 * (nlength\ (Suc\ m))^2))$
shows *transforms tm26 tps0 ttt tps26*
unfolding *tm26-def*
proof (*tform*)
show *61* < *length tps25*
using *lentpsF tps25 k* **by** (*simp-all only*: *length-list-update*)
have *tps25* ! *61* = *tpsF* ! *61*
using *tps25* **by** (*simp only*: *nth-list-update-neq*)
then show *tps25* ! *61* = (⌊*0*⌋$_N$, *1*)
using *tpsF k canrepr-0* **by** *simp*
show *tps26* = *tps25*[*61* := (⌊*H*⌋$_N$, *1*)]
unfolding *tps26-def tps25* **by** (*simp only*:)

```
  show ttt = 203 + (2 * d-polynomial p + 826) * (H + m') ^ 4 + 3 * nlength m' + 11 * nlength H +
      5627 * H ^ 4 * (3 + nlength (3 * H + m' * H))^2 + 1875 * H ^ 4 +
      15 * nlength n + 3764 * H ^ 4 * (3 + nlength (3 * H + 2 * n * H))^2 +
      Suc n * (9 + 1897 * (H ^ 4 * (nlength (1 + 2 * n))^2)) + 6 * nlength (2 * n + 3) +
      5 * nlength m + Suc (p n) * (9 + 1897 * (H ^ 4 * (nlength (Suc m))^2)) +
      (10 + 2 * nlength 0 + 2 * nlength H)
    using assms by simp
qed

definition tps27 ≡ tpsF
  [11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊m⌋_N, 1),
   17 := (⌊T'⌋_N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
   18 := (⌊m'⌋_N, 1),
   19 := (⌊H⌋_N, 1),
   20 := (⌊m' * H⌋_N, 1),
   1 := nlltape (formula-n Φ_0 @ formula-n Φ_1 @ formula-n Φ_2 @ formula-n Φ_3 @ formula-n Φ_4),
   61 := (⌊H⌋_N, 1),
   60 := (⌊m⌋_N, 1)]

lemma tm27 [transforms-intros]:
  assumes ttt = 227 + (2 * d-polynomial p + 826) * (H + m') ^ 4 + 3 * nlength m' + 13 * nlength H +
    5627 * H ^ 4 * (3 + nlength (3 * H + m' * H))^2 + 1875 * H ^ 4 +
    15 * nlength n + 3764 * H ^ 4 * (3 + nlength (3 * H + 2 * n * H))^2 +
    Suc n * (9 + 1897 * (H ^ 4 * (nlength (1 + 2 * n))^2)) + 6 * nlength (2 * n + 3) +
    8 * nlength m + Suc (p n) * (9 + 1897 * (H ^ 4 * (nlength (Suc m))^2))
  shows transforms tm27 tps0 ttt tps27
  unfolding tm27-def
proof (tform)
  show 16 < length tps26 60 < length tps26
    using lentpsF tps26-def k by (simp-all only: length-list-update)
  show tps26 ! 16 = (⌊m⌋_N, 1)
    using tps26-def k lentpsF by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  have tps26 ! 60 = tpsF ! 60
    using tps26-def by (simp only: nth-list-update-neq)
  then show tps26 ! 60 = (⌊0⌋_N, 1)
    using tpsF k canrepr-0 by simp
  show tps27 = tps26[60 := (⌊m⌋_N, 1)]
    unfolding tps27-def tps26-def by (simp only:)
  show ttt = 213 + (2 * d-polynomial p + 826) * (H + m') ^ 4 + 3 * nlength m' + 13 * nlength H +
      5627 * H ^ 4 * (3 + nlength (3 * H + m' * H))^2 + 1875 * H ^ 4 +
      15 * nlength n + 3764 * H ^ 4 * (3 + nlength (3 * H + 2 * n * H))^2 +
      Suc n * (9 + 1897 * (H ^ 4 * (nlength (1 + 2 * n))^2)) + 6 * nlength (2 * n + 3) +
      5 * nlength m + Suc (p n) * (9 + 1897 * (H ^ 4 * (nlength (Suc m))^2)) +
      (14 + 3 * (nlength m + nlength 0))
    using assms by simp
qed

definition tps28 ≡ tpsF
  [11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊m⌋_N, 1),
   17 := (⌊T'⌋_N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
   18 := (⌊m'⌋_N, 1),
   19 := (⌊H⌋_N, 1),
   20 := (⌊m' * H⌋_N, 1),
   1 := nlltape (formula-n Φ_0 @ formula-n Φ_1 @ formula-n Φ_2 @ formula-n Φ_3 @ formula-n Φ_4),
   61 := (⌊H⌋_N, 1),
```

60 := (⌊*Suc m*⌋$_N$, *1*)]

lemma *tm28* [*transforms-intros*]:
assumes *ttt* = *232* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *13* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
15 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2)) + *6* ∗ *nlength* (*2* ∗ *n* + *3*) +
10 ∗ *nlength m* + *Suc* (*p n*) ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*Suc m*))^2))
shows *transforms tm28 tps0 ttt tps28*
unfolding *tm28-def*
proof (*tform*)
show *60* < *length tps27*
using *lentpsF tps27-def k* **by** (*simp-all only*: *length-list-update*)
show *tps27* ! *60* = (⌊*m*⌋$_N$, *1*)
using *tps27-def k lentpsF* **by** (*simp only*: *length-list-update nth-list-update-eq*)
show *tps28* = *tps27*[*60* := (⌊*Suc m*⌋$_N$, *1*)]
unfolding *tps28-def tps27-def* **by** (*simp only*: *list-update-overwrite*)
show *ttt* = *227* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *13* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
15 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2)) + *6* ∗ *nlength* (*2* ∗ *n* + *3*) +
8 ∗ *nlength m* + *Suc* (*p n*) ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*Suc m*))^2)) +
(*5* + *2* ∗ *nlength m*)
using *assms* **by** *simp*
qed

definition *tps29* ≡ *tpsF*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t. exc zs t* <#> *0*) [*0..*<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t. exc zs t* <#> *1*) [*0..*<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape* (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4),
61 := (⌊*H*⌋$_N$, *1*),
60 := (⌊*Suc m*⌋$_N$, *1*),
68 := (⌊*Suc m*⌋$_N$, *1*)]

lemma *tm29* [*transforms-intros*]:
assumes *ttt* = *246* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *13* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
15 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2)) + *6* ∗ *nlength* (*2* ∗ *n* + *3*) +
10 ∗ *nlength m* + *Suc* (*p n*) ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*Suc m*))^2)) +
3 ∗ *nlength* (*Suc m*)
shows *transforms tm29 tps0 ttt tps29*
unfolding *tm29-def*
proof (*tform*)
show *60* < *length tps28 68* < *length tps28*
using *lentpsF tps28-def k* **by** (*simp-all only*: *length-list-update*)
show *tps28* ! *60* = (⌊*Suc m*⌋$_N$, *1*)
using *tps28-def k lentpsF* **by** (*simp only*: *length-list-update nth-list-update-eq*)
have *tps28* ! *68* = *tpsF* ! *68*
using *tps28-def* **by** (*simp only*: *nth-list-update-neq*)
then show *tps28* ! *68* = (⌊*0*⌋$_N$, *1*)
using *tpsF k canrepr-0* **by** *simp*
show *tps29* = *tps28*[*68* := (⌊*Suc m*⌋$_N$, *1*)]
unfolding *tps29-def tps28-def* **by** (*simp only*: *list-update-overwrite*)
show *ttt* = *232* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m'*) ^ *4* + *3* ∗ *nlength m'* + *13* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m'* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +

15 * nlength n + 3764 * H ^ 4 * (3 + nlength (3 * H + 2 * n * H))^2 +
Suc n * (9 + 1897 * (H ^ 4 * (nlength (1 + 2 * n))^2)) + 6 * nlength (2 * n + 3) +
10 * nlength m + Suc (p n) * (9 + 1897 * (H ^ 4 * (nlength (Suc m))^2)) +
(14 + 3 * (nlength (Suc m) + nlength 0))
using *assms* **by** *simp*
qed

definition *tps30* ≡ *tpsF*
[11 := ($\lfloor n \rfloor_N$, 1),
15 := ($\lfloor p\ n \rfloor_N$, 1),
16 := ($\lfloor m \rfloor_N$, 1),
17 := ($\lfloor T' \rfloor_N$, 1),
4 := ($\lfloor$map (λt. exc zs t <#> 0) [0..<Suc T']$\rfloor_{NL}$, 1),
7 := ($\lfloor$map (λt. exc zs t <#> 1) [0..<Suc T']$\rfloor_{NL}$, 1),
18 := ($\lfloor m' \rfloor_N$, 1),
19 := ($\lfloor H \rfloor_N$, 1),
20 := ($\lfloor m' * H \rfloor_N$, 1),
1 := nlltape (formula-n Φ_0 @ formula-n Φ_1 @ formula-n Φ_2 @ formula-n Φ_3 @ formula-n Φ_4),
61 := ($\lfloor H \rfloor_N$, 1),
60 := ($\lfloor Suc\ m \rfloor_N$, 1),
68 := ($\lfloor T' + Suc\ m \rfloor_N$, 1)]

lemma *tm30* [*transforms-intros*]:
assumes *ttt* = 256 + (2 * d-polynomial p + 826) * (H + m') ^ 4 + 3 * nlength m' + 13 * nlength H +
5627 * H ^ 4 * (3 + nlength (3 * H + m' * H))^2 + 1875 * H ^ 4 +
15 * nlength n + 3764 * H ^ 4 * (3 + nlength (3 * H + 2 * n * H))^2 +
Suc n * (9 + 1897 * (H ^ 4 * (nlength (1 + 2 * n))^2)) + 6 * nlength (2 * n + 3) +
10 * nlength m + Suc (p n) * (9 + 1897 * (H ^ 4 * (nlength (Suc m))^2)) +
3 * nlength (Suc m) + 3 * max (nlength T') (nlength (Suc m))
shows *transforms tm30 tps0 ttt tps30*
unfolding *tm30-def*
proof (*tform*)
show 17 < length tps29 68 < length tps29
using *lentpsF tps29-def k* **by** (*simp-all only*: *length-list-update*)
show tps29 ! 17 = ($\lfloor T' \rfloor_N$, 1)
using *tps29-def k lentpsF* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show tps29 ! 68 = ($\lfloor Suc\ m \rfloor_N$, 1)
using *tps29-def k lentpsF* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show tps30 = tps29[68 := ($\lfloor T' + Suc\ m \rfloor_N$, 1)]
unfolding *tps30-def tps29-def* **by** (*simp only*: *list-update-overwrite*)
show ttt = 246 + (2 * d-polynomial p + 826) * (H + m') ^ 4 + 3 * nlength m' + 13 * nlength H +
5627 * H ^ 4 * (3 + nlength (3 * H + m' * H))^2 + 1875 * H ^ 4 +
15 * nlength n + 3764 * H ^ 4 * (3 + nlength (3 * H + 2 * n * H))^2 +
Suc n * (9 + 1897 * (H ^ 4 * (nlength (1 + 2 * n))^2)) + 6 * nlength (2 * n + 3) +
10 * nlength m + Suc (p n) * (9 + 1897 * (H ^ 4 * (nlength (Suc m))^2)) +
3 * nlength (Suc m) + (3 * max (nlength T') (nlength (Suc m)) + 10)
using *assms* **by** *simp*
qed

definition *tps31* ≡ *tpsF*
[11 := ($\lfloor n \rfloor_N$, 1),
15 := ($\lfloor p\ n \rfloor_N$, 1),
16 := ($\lfloor m \rfloor_N$, 1),
17 := ($\lfloor T' \rfloor_N$, 1),
4 := ($\lfloor$map (λt. exc zs t <#> 0) [0..<Suc T']$\rfloor_{NL}$, 1),
7 := ($\lfloor$map (λt. exc zs t <#> 1) [0..<Suc T']$\rfloor_{NL}$, 1),
18 := ($\lfloor m' \rfloor_N$, 1),
19 := ($\lfloor H \rfloor_N$, 1),
20 := ($\lfloor m' * H \rfloor_N$, 1),
1 := nlltape
(formula-n Φ_0 @ formula-n Φ_1 @ formula-n Φ_2 @ formula-n Φ_3 @ formula-n Φ_4 @
formula-n Φ_5),
61 := ($\lfloor H \rfloor_N$, 1),

$60 := (\lfloor Suc\ m \rfloor_N,\ 1),$
$68 := (\lfloor T' + Suc\ m \rfloor_N,\ 1),$
$60 := (\lfloor Suc\ m + T' \rfloor_N,\ 1),$
$60 + 3 := (\lfloor 1 \rfloor_N,\ 1)]$

definition $ttt31 \equiv 256 + (2 * d\text{-}polynomial\ p + 826) * (H + m')\ \hat{}\ 4 + 3 * nlength\ m' + 13 * nlength\ H + 5627 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + m' * H))^2 + 1875 * H\ \hat{}\ 4 + 15 * nlength\ n + 3764 * H\ \hat{}\ 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 + Suc\ n * (9 + 1897 * (H\ \hat{}\ 4 * (nlength\ (1 + 2 * n))^2)) + 6 * nlength\ (2 * n + 3) + 10 * nlength\ m + Suc\ (p\ n) * (9 + 1897 * (H\ \hat{}\ 4 * (nlength\ (Suc\ m))^2)) + 3 * nlength\ (Suc\ m) + 3 * max\ (nlength\ T')\ (nlength\ (Suc\ m)) + Suc\ T' * (9 + 1891 * (H\ \hat{}\ 4 * (nlength\ (Suc\ m + T'))^2))$

lemma *le-N*: $y \le 2 * n + 2 * p\ n + 3 + T' \Longrightarrow y \le N$
using *H-gr-2 m'-def N-eq*
by (*metis Suc-1 Suc-leI add-2-eq-Suc' add-leE le-trans mult-le-mono1 nat-mult-1*)

lemma *n-le-N*: $n \le N$
using *le-N* **by** *simp*

lemma *H-le-N*: $H \le N$
using *N-eq* **by** *simp*

lemma *N-ge-1*: $N \ge 1$
using *H-le-N H-ge-3* **by** *simp*

lemma *pow2-sum-le*:
fixes $a\ b :: nat$
shows $(a + b)\ \hat{}\ 2 \le a\ \hat{}\ 2 + (2 * a + 1) * b\ \hat{}\ 2$
proof −
have $(a + b)\ \hat{}\ 2 = a\ \hat{}\ 2 + 2 * a * b + b\ \hat{}\ 2$
by *algebra*
also have $... \le a\ \hat{}\ 2 + 2 * a * b\ \hat{}\ 2 + b\ \hat{}\ 2$
by (*simp add*: *power2-nat-le-imp-le*)
also have $... = a\ \hat{}\ 2 + (2 * a + 1) * b\ \hat{}\ 2$
by *simp*
finally show *?thesis* **.**
qed

lemma *ttt31*: $ttt31 \le (32 * d\text{-}polynomial\ p + 222011) * H\ \hat{}\ 4 * N\ \hat{}\ 4$
proof −
have *a*: $Suc\ T' * (9 + 1891 * (H\ \hat{}\ 4 * (nlength\ (Suc\ m + T'))^2)) \le 1900 * H\ \hat{}\ 4 * N\ \hat{}\ 4$
proof −
have $Suc\ (2 * n + 2 * p\ n + 2) + T' \le 2 * n + 2 * p\ n + 3 + T'$
by *simp*
also have $... \le H * (2 * n + 2 * p\ n + 3 + T')$
using *H-gr-2* **by** *simp*
finally have $Suc\ m + T' \le N$
unfolding *N-eq m-def* **by** *simp*
then have $nlength\ (Suc\ m + T') \le N$
using *nlength-le-n order.trans* **by** *auto*
then have $(nlength\ (Suc\ m + T'))\ \hat{}\ 2 \le N\ \hat{}\ 2$
by *simp*
then have $H\ \hat{}\ 4 * (nlength\ (Suc\ m + T'))\ \hat{}\ 2 \le H\ \hat{}\ 4 * N\ \hat{}\ 2$
using *mult-le-mono* **by** *simp*
then have $9 + 1891 * (H\ \hat{}\ 4 * (nlength\ (Suc\ m + T'))^2) \le 9 + 1891 * H\ \hat{}\ 4 * N\ \hat{}\ 2$
by *simp*
then have $Suc\ T' * (9 + 1891 * (H\ \hat{}\ 4 * (nlength\ (Suc\ m + T'))^2)) \le Suc\ T' * (9 + 1891 * H\ \hat{}\ 4 * N\ \hat{}\ 2)$
using *mult-le-mono2* **by** *blast*
also have $... \le N * (9 + 1891 * H\ \hat{}\ 4 * N\ \hat{}\ 2)$
proof −
have $Suc\ T' \le N$

using *le-N* **by** *simp*
then show *?thesis*
using *mult-le-mono1* **by** *blast*
qed
also have ... $= 9 * N + 1891 * H \char`^ 4 * N \char`^ 3$
by *algebra*
also have ... $\leq 9 * N \char`^ 3 + 1891 * H \char`^ 4 * N \char`^ 3$
using *linear-le-pow* **by** *simp*
also have ... $\leq 9 * N \char`^ 3 + 1891 * H \char`^ 4 * N \char`^ 4$
using *pow-mono'*[*of 3 4*] **by** *simp*
also have ... $\leq 9 * N \char`^ 4 + 1891 * H \char`^ 4 * N \char`^ 4$
using *pow-mono'*[*of 3 4*] **by** *simp*
also have ... $\leq 9 * H \char`^ 4 * N \char`^ 4 + 1891 * H \char`^ 4 * N \char`^ 4$
using *H-ge-3* **by** *simp*
also have ... $= 1900 * H \char`^ 4 * N \char`^ 4$
by *simp*
finally show *?thesis* **.**
qed

have *b*: $5627 * H \char`^ 4 * (3 + nlength\ (3 * H + m' * H))^2 \leq 140675 * H \char`^ 4 * N \char`^ 4$
proof −
have $3 * H + m' * H \leq 2 * N$
using *N-eq m'-def* **by** *simp*
then have $nlength\ (3 * H + m' * H) \leq nlength\ (2 * N)$
using *nlength-mono* **by** *simp*
also have ... $\leq Suc\ (nlength\ N)$
using *le-trans nlength-times2* **by** *blast*
also have ... $\leq Suc\ N$
using *nlength-le-n* **by** *simp*
finally have $nlength\ (3 * H + m' * H) \leq Suc\ N$ **.**
then have $3 + nlength\ (3 * H + m' * H) \leq 4 + N$
by *simp*
then have $(3 + nlength\ (3 * H + m' * H))\ \char`^ 2 \leq (4 + N)\ \char`^ 2$
by *simp*
also have ... $\leq 16 + 9 * N \char`^ 2$
using *pow2-sum-le*[*of 4 N*] **by** *simp*
finally have $(3 + nlength\ (3 * H + m' * H))\ \char`^ 2 \leq 16 + 9 * N \char`^ 2$ **.**
then have $5627 * H \char`^ 4 * (3 + nlength\ (3 * H + m' * H))^2 \leq 5627 * H \char`^ 4 * (16 + 9 * N \char`^ 2)$
by *simp*
also have ... $= 5627 * H \char`^ 4 * 16 + 5627 * H \char`^ 4 * 9 * N \char`^ 2$
by *algebra*
also have ... $= 90032 * H \char`^ 4 + 50643 * H \char`^ 4 * N \char`^ 2$
by *simp*
also have ... $\leq 90032 * H \char`^ 4 * N \char`^ 2 + 50643 * H \char`^ 4 * N \char`^ 2$
using *pow-mono' N-ge-1* **by** *simp*
also have ... $= 140675 * H \char`^ 4 * N \char`^ 2$
by *simp*
also have ... $\leq 140675 * H \char`^ 4 * N \char`^ 4$
using *pow-mono'* **by** *simp*
finally show *?thesis* **.**
qed

have *c*: $3764 * H \char`^ 4 * (3 + nlength\ (3 * H + 2 * n * H))^2 \leq 60224 * H \char`^ 4 * N \char`^ 4$
proof −
have $3 * H \leq N$ **and** $2 * n * H \leq N$
using *N-eq* **by** *simp-all*
then have $3 * H + 2 * n * H \leq 2 * N$
by *simp*
then have $nlength\ (3 * H + 2 * n * H) \leq 3 * H + 2 * n * H$
using *nlength-le-n* **by** *simp*
then have $nlength\ (3 * H + 2 * n * H) \leq H * (3 + 2 * n)$
by (*metis distrib-right mult.commute*)
also have ... $\leq N$

using *N-eq* **by** *simp*
finally have *nlength* $(3 * H + 2 * n * H) \leq N$ **.**
then have $3 +$ *nlength* $(3 * H + 2 * n * H) \leq 3 + N$
by *simp*
then have $(3 +$ *nlength* $(3 * H + 2 * n * H))^2 \leq (3 + N)$ ^ 2
by *simp*
also have ... $\leq 9 + 7 * N$ ^ 2
using *pow2-sum-le*[*of 3 N*] **by** *simp*
finally have $(3 +$ *nlength* $(3 * H + 2 * n * H))^2 \leq 9 + 7 * N$ ^ 2 **.**
then have $3764 * H$ ^ $4 * (3 +$ *nlength* $(3 * H + 2 * n * H))^2 \leq 3764 * H$ ^ $4 * (9 + 7 * N$ ^ $2)$
by *simp*
also have ... $= 3764 * H$ ^ $4 * 9 + 3764 * H$ ^ $4 * 7 * N$ ^ 2
by *algebra*
also have ... $= 33876 * H$ ^ $4 + 26348 * H$ ^ $4 * N$ ^ 2
by *simp*
also have ... $\leq 33876 * H$ ^ $4 * N$ ^ $2 + 26348 * H$ ^ $4 * N$ ^ 2
using *pow-mono′ N-ge-1* **by** *simp*
also have ... $= 60224 * H$ ^ $4 * N$ ^ 2
by *simp*
also have ... $\leq 60224 * H$ ^ $4 * N$ ^ 4
using *pow-mono′* **by** *simp*
finally show *?thesis* **.**
qed

have *d*: *Suc* $(p\ n) * (9 + 1897 * (H$ ^ $4 * ($*nlength* $(Suc\ m))^2)) \leq 1906 * H$ ^ $4 * N$ ^ 4
proof −
have *Suc* $(p\ n) \leq N$
using *le-N* **by** *simp*
then have *Suc* $(p\ n) * (9 + 1897 * (H$ ^ $4 * ($*nlength* $(Suc\ m))^2)) \leq N * (9 + 1897 * (H$ ^ $4 * ($*nlength* $(Suc\ m))^2))$
using *mult-le-mono1* **by** *blast*
also have ... $\leq N * (9 + 1897 * (H$ ^ $4 * ($*nlength* $N)^2))$
proof −
have *Suc* $m \leq N$
using *m-def le-N* **by** *simp*
then show *?thesis*
using *H4-nlength H-ge-3 add-le-mono less-or-eq-imp-le mult-le-mono* **by** *presburger*
qed
also have ... $\leq N * (9 + 1897 * (H$ ^ $4 * N^2))$
using *nlength-le-n* **by** *simp*
also have ... $= N * 9 + N * 1897 * H$ ^ $4 * N$ ^ 2
by (*simp add*: *add-mult-distrib2*)
also have ... $\leq N$ ^ $3 * 9 + N * 1897 * H$ ^ $4 * N$ ^ 2
using *linear-le-pow* **by** *simp*
also have ... $\leq 9 * H$ ^ $4 * N$ ^ $3 + N * 1897 * H$ ^ $4 * N$ ^ 2
using *H-ge-3* **by** *simp*
also have ... $= 9 * H$ ^ $4 * N$ ^ $3 + 1897 * H$ ^ $4 * N$ ^ 3
by *algebra*
also have ... $= 1906 * H$ ^ $4 * N$ ^ 3
by *simp*
also have ... $\leq 1906 * H$ ^ $4 * N$ ^ 4
using *pow-mono′* **by** *simp*
finally show *?thesis* **.**
qed

have *e*: *Suc* $n * (9 + 1897 * (H$ ^ $4 * ($*nlength* $(1 + 2 * n))^2)) \leq 1906 * H$ ^ $4 * N$ ^ 4
proof −
have *nlength* $(1 + 2 * n) \leq N$
using *le-N nlength-le-n*[*of* $1 + 2 * n$] **by** *simp*
then have $($*nlength* $(1 + 2 * n))$ ^ $2 \leq N$ ^ 2
by *simp*
then have *Suc* $n * (9 + 1897 * (H$ ^ $4 * ($*nlength* $(1 + 2 * n))^2)) \leq$ *Suc* $n * (9 + 1897 * (H$ ^ $4 * N^2))$
using *add-le-mono less-or-eq-imp-le mult-le-mono2* **by** *presburger*

also have ... $=$ *Suc n* $*$ (*9* $+$ *1897* $*$ *H* ^ *4* $*$ N^2)
  by *simp*
also have ... $\leq$ *N* $*$ (*9* $+$ *1897* $*$ *H* ^ *4* $*$ *N* ^ *2*)
  using *mult-le-mono1*[*OF le-N*[*of Suc n*]] **by** *simp*
also have ... $=$ *N* $*$ *9* $+$ *1897* $*$ *H* ^ *4* $*$ *N* ^ *3*
  by *algebra*
also have ... $\leq$ *9* $*$ *N* ^ *3* $+$ *1897* $*$ *H* ^ *4* $*$ *N* ^ *3*
  using *linear-le-pow* **by** *simp*
also have ... $\leq$ *9* $*$ *H* ^ *4* $*$ *N* ^ *3* $+$ *1897* $*$ *H* ^ *4* $*$ *N* ^ *3*
  using *H-ge-3* **by** *simp*
also have ... $=$ *1906* $*$ *H* ^ *4* $*$ *N* ^ *3*
  by *simp*
also have ... $\leq$ *1906* $*$ *H* ^ *4* $*$ *N* ^ *4*
  using *pow-mono'* **by** *simp*
finally show *?thesis* **.**
qed

have *nlength-le-GN*: *y* $\leq$ *N* $\Longrightarrow$ *nlength y* $\leq$ *H* ^ *4* $*$ *N* ^ *4* **for** *y*
proof $-$
  assume *y* $\leq$ *N*
  then have *nlength y* $\leq$ *N* ^ *4*
    using *nlength-le-n linear-le-pow H-ge-3* **by** (*meson dual-order.trans zero-less-numeral*)
  also have ... $\leq$ *H* ^ *4* $*$ *N* ^ *4*
    using *H-gr-2* **by** *simp*
  finally show *?thesis* **.**
qed

have *f*: *13* $*$ *nlength H* $\leq$ *13* $*$ *H* ^ *4* $*$ *N* ^ *4*
  using *nlength-le-GN*[*of H*] *N-eq* **by** *simp*
have *g*: *15* $*$ *nlength n* $\leq$ *15* $*$ *H* ^ *4* $*$ *N* ^ *4*
  using *nlength-le-GN*[*OF n-le-N*] **by** *simp*
have *h*: *6* $*$ *nlength* (*2* $*$ *n* $+$ *3*) $\leq$ *6* $*$ *H* ^ *4* $*$ *N* ^ *4*
  using *nlength-le-GN le-N* **by** *simp*
have *i*: *10* $*$ *nlength m* $\leq$ *10* $*$ *H* ^ *4* $*$ *N* ^ *4*
  using *m-def nlength-le-GN le-N* **by** *simp*
have *j*: *3* $*$ *nlength* (*Suc m*) $\leq$ *3* $*$ *H* ^ *4* $*$ *N* ^ *4*
  using *m-def nlength-le-GN le-N* **by** *simp*
have *k*: *3* $*$ *max* (*nlength T'*) (*nlength* (*Suc m*)) $\leq$ *3* $*$ *H* ^ *4* $*$ *N* ^ *4*
proof $-$
  have *nlength T'* $\leq$ *H* ^ *4* $*$ *N* ^ *4*
    using *nlength-le-GN le-N* **by** *simp*
  moreover have *nlength* (*Suc m*) $\leq$ *H* ^ *4* $*$ *N* ^ *4*
    using *m-def nlength-le-GN le-N* **by** *simp*
  ultimately show *?thesis*
    by *simp*
qed
have *l*: *1875* $*$ *H* ^ *4* $\leq$ *1875* $*$ *H* ^ *4* $*$ *N* ^ *4*
  using *N-ge-1* **by** *simp*
have *m*: *3* $*$ *nlength m'* $\leq$ *3* $*$ *H* ^ *4* $*$ *N* ^ *4*
  using *m'-def le-N le-refl nat-mult-le-cancel-disj nlength-le-GN* **by** *simp*

have *ttt31* $\leq$ *256* $+$ (*2* $*$ *d-polynomial p* $+$ *826*) $*$ (*H* $+$ *m'*) ^ *4* $+$ *1928* $*$ *H* ^ *4* $*$ *N* ^ *4* $+$
  140675 $*$ *H* ^ *4* $*$ *N* ^ *4* $+$ *60224* $*$ *H* ^ *4* $*$ *N* ^ *4* $+$ *1906* $*$ *H* ^ *4* $*$ *N* ^ *4* $+$ *1906* $*$ *H* ^ *4* $*$ *N* ^ *4* $+$
1900 $*$ *H* ^ *4* $*$ *N* ^ *4*
  using *ttt31-def a b c d e m l k f h i g j* **by** *linarith*
also have ... $=$ *256* $+$ (*2* $*$ *d-polynomial p* $+$ *826*) $*$ (*H* $+$ *m'*) ^ *4* $+$ *208539* $*$ *H* ^ *4* $*$ *N* ^ *4*
  by *simp*
also have ... $\leq$ *256* $+$ (*2* $*$ *d-polynomial p* $+$ *826*) $*$ (*N* $+$ *N*) ^ *4* $+$ *208539* $*$ *H* ^ *4* $*$ *N* ^ *4*
proof $-$
  have *H* $\leq$ *N*
    using *N-eq* **by** *simp*
  then show *?thesis*
    using *le-N*[*of m'*] *m'-def* **by** *simp*

qed
also have ... = *256 + (2 ∗ d-polynomial p + 826) ∗ (2 ∗ N) ^ 4 + 208539 ∗ H ^ 4 ∗ N ^ 4*
by *algebra*
also have ... = *256 + (2 ∗ d-polynomial p + 826) ∗ 16 ∗ N ^ 4 + 208539 ∗ H ^ 4 ∗ N ^ 4*
by *simp*
also have ... ≤ *256 + (2 ∗ d-polynomial p + 826) ∗ 16 ∗ H ^ 4 ∗ N ^ 4 + 208539 ∗ H ^ 4 ∗ N ^ 4*
using *H-ge-3* **by** *simp*
also have ... = *256 + (32 ∗ d-polynomial p + 13216) ∗ H ^ 4 ∗ N ^ 4 + 208539 ∗ H ^ 4 ∗ N ^ 4*
by *simp*
also have ... = *256 + (32 ∗ d-polynomial p + 221755) ∗ H ^ 4 ∗ N ^ 4*
by *algebra*
also have ... ≤ *256 ∗ H ^ 4 + (32 ∗ d-polynomial p + 221755) ∗ H ^ 4 ∗ N ^ 4*
using *H-gr-2* **by** *simp*
also have ... ≤ *256 ∗ H ^ 4 ∗ N ^ 4 + (32 ∗ d-polynomial p + 221755) ∗ H ^ 4 ∗ N ^ 4*
using *N-ge-1* **by** *simp*
also have ... = *(32 ∗ d-polynomial p + 222011) ∗ H ^ 4 ∗ N ^ 4*
by *algebra*
finally show *?thesis* **.**
qed

lemma *tm31* [*transforms-intros*]: *transforms tm31 tps0 ttt31 tps31*
unfolding *tm31-def*
proof (*tform transforms-intros*: *tm-PHI5*)
show *60 + 8 < length tps30*
using *lentpsF tps30-def k* **by** (*simp-all only*: *length-list-update*)
show *tps30 ! 1 = nlltape (formula-n* Φ_0 *@ formula-n* Φ_1 *@ formula-n* Φ_2 *@ formula-n* Φ_3 *@ formula-n* Φ_4*)*
using *tps30-def k lentpsF* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have ∗: *2 ∗ n + 2 ∗ p n + 3 = Suc m*
using *m-def One-nat-def Suc-1 add-Suc-right numeral-3-eq-3* **by** *presburger*
have *tps30 ! 60 =* (⌊*Suc m*⌋$_N$, *1*)
using *tps30-def k lentpsF* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show *tps30 ! 60 =* (⌊*2 ∗ n + 2 ∗ p n + 3*⌋$_N$, *1*)
using ∗ **by** *presburger*
have *tps30 ! 61 =* (⌊*H*⌋$_N$, *1*)
using *tps30-def k lentpsF* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show *tps30 ! (60 + 1) =* (⌊*H*⌋$_N$, *1*)
by *simp*
have *tps30 ! 62 = tpsF ! 62*
using *tps30-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps30 ! (60 + 2) =* (⌊*0*⌋$_N$, *1*)
using *tpsF k canrepr-0* **by** *simp*
have *tps30 ! 63 = tpsF ! 63*
using *tps30-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps30 ! (60 + 3) =* (⌊[]⌋, *1*)
using *tpsF k* **by** *simp*
have *tps30 ! 64 = tpsF ! 64*
using *tps30-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps30 ! (60 + 4) =* (⌊[]⌋, *1*)
using *tpsF k* **by** *simp*
have *tps30 ! 65 = tpsF ! 65*
using *tps30-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps30 ! (60 + 5) =* (⌊[]⌋, *1*)
using *tpsF k* **by** *simp*
have *tps30 ! 66 = tpsF ! 66*
using *tps30-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps30 ! (60 + 6) =* (⌊[]⌋, *1*)
using *tpsF k* **by** *simp*
have *tps30 ! 67 = tpsF ! 67*
using *tps30-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps30 ! (60 + 7) =* (⌊[]⌋, *1*)
using *tpsF k* **by** *simp*
have *tps30 ! 68 =* (⌊*T' + Suc m*⌋$_N$, *1*)
using *tps30-def k lentpsF* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)

then have *tps30* ! (*60* + *8*) = (⌊*Suc m* + *1* ∗ *T′*⌋$_N$, *1*)
by (*metis add.commute add-One-commute nat-mult-1 numeral-plus-numeral semiring-norm*(*2*) *semiring-norm*(*4*) *semiring-norm*(*6*) *semiring-norm*(*7*))
then show *tps30* ! (*60* + *8*) = (⌊*2* ∗ *n* + *2* ∗ *p n* + *3* + *T′*⌋$_N$, *1*)
using ∗ *nat-mult-1* **by** *presburger*
have *tps31* = *tps30*
[*60* := (⌊*Suc m* + *T′*⌋$_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5),
60 + *3* := (⌊*1*⌋$_N$, *1*)]
unfolding *tps31-def tps30-def* **by** (*simp only*: *list-update-swap list-update-overwrite*)
then show *tps31* = *tps30*
[*60* := (⌊*2* ∗ *n* + *2* ∗ *p n* + *3* + *T′*⌋$_N$, *1*),
1 := *nlltape*
((*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4) @
formula-n Φ_5),
60 + *3* := (⌊*1*⌋$_N$, *1*)]
using ∗ **by** (*metis append-eq-appendI*)
show *ttt31* = *256* + (*2* ∗ *d-polynomial p* + *826*) ∗ (*H* + *m′*) ^ *4* + *3* ∗ *nlength m′* + *13* ∗ *nlength H* +
5627 ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *m′* ∗ *H*))^2 + *1875* ∗ *H* ^ *4* +
15 ∗ *nlength n* + *3764* ∗ *H* ^ *4* ∗ (*3* + *nlength* (*3* ∗ *H* + *2* ∗ *n* ∗ *H*))^2 +
Suc n ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*1* + *2* ∗ *n*))^2)) + *6* ∗ *nlength* (*2* ∗ *n* + *3*) +
10 ∗ *nlength m* + *Suc* (*p n*) ∗ (*9* + *1897* ∗ (*H* ^ *4* ∗ (*nlength* (*Suc m*))^2)) +
3 ∗ *nlength* (*Suc m*) + *3* ∗ *max* (*nlength T′*) (*nlength* (*Suc m*)) +
Suc T′ ∗ (*9* + *1891* ∗ (*H* ^ *4* ∗ (*nlength* (*2* ∗ *n* + *2* ∗ *p n* + *3* + *T′*))^2))
using *ttt31-def m-def One-nat-def Suc-1 add-Suc-right numeral-3-eq-3* **by** *presburger*
qed

definition *tpsG* ≡ *tpsF*
[*61* := (⌊*H*⌋$_N$, *1*),
60 := (⌊*Suc m*⌋$_N$, *1*),
68 := (⌊*T′* + *Suc m*⌋$_N$, *1*),
60 := (⌊*Suc m* + *T′*⌋$_N$, *1*),
60 + *3* := (⌊*1*⌋$_N$, *1*)]

lemma *tpsG*: *68* < *j* ⟹ *j* < *110* ⟹ *tpsG* ! *j* = (⌊[]⌋, *1*)
using *tpsG-def tpsF* **by** *simp*

lemma *lentpsG*: *length tpsG* = *110*
using *lentpsF tpsG-def* **by** *simp*

lemma *tps31*: *tps31* = *tpsG*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T′*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
18 := (⌊*m′*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m′* ∗ *H*⌋$_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5)]
unfolding *tps31-def tpsG-def* **by** (*simp only*: *list-update-swap*)

definition *tps32* ≡ *tpsG*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T′*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),

```
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋NL, 1),
   18 := (⌊m'⌋N, 1),
   19 := (⌊H⌋N, 1),
   20 := (⌊m' * H⌋N, 1),
   1 := nlltape
         (formula-n Φ0 @ formula-n Φ1 @ formula-n Φ2 @ formula-n Φ3 @ formula-n Φ4 @
          formula-n Φ5),
   69 := (⌊2⌋N, 1)]

lemma tm32 [transforms-intros]:
  assumes ttt = ttt31 + 14
  shows transforms tm32 tps0 ttt tps32
  unfolding tm32-def
proof (tform)
  show 69 < length tps31
    using lentpsF tps31-def k by (simp-all only: length-list-update)
  have tps31 ! 69 = tpsF ! 69
    using tps31-def by (simp only: nth-list-update-neq)
  then show tps31 ! 69 = (⌊0⌋N, 1)
    using tpsF k canrepr-0 by simp
  show tps32 = tps31[69 := (⌊2⌋N, 1)]
    unfolding tps32-def tps31 by (simp only:)
  show ttt = ttt31 + (10 + 2 * nlength 0 + 2 * nlength 2)
    using assms nlength-2 by simp
qed

definition tps33 ≡ tpsG
  [11 := (⌊n⌋N, 1),
   15 := (⌊p n⌋N, 1),
   16 := (⌊m⌋N, 1),
   17 := (⌊T'⌋N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋NL, 1),
   18 := (⌊m'⌋N, 1),
   19 := (⌊H⌋N, 1),
   20 := (⌊m' * H⌋N, 1),
   1 := nlltape
         (formula-n Φ0 @ formula-n Φ1 @ formula-n Φ2 @ formula-n Φ3 @ formula-n Φ4 @
          formula-n Φ5),
   69 := (⌊2⌋N, 1),
   70 := (⌊H⌋N, 1)]

lemma tm33 [transforms-intros]:
  assumes ttt = ttt31 + 24 + 2 * nlength H
  shows transforms tm33 tps0 ttt tps33
  unfolding tm33-def
proof (tform)
  show 70 < length tps32
    using lentpsG tps32-def k by (simp-all only: length-list-update)
  have tps32 ! 70 = tpsG ! 70
    using tps32-def by (simp only: nth-list-update-neq)
  then show tps32 ! 70 = (⌊0⌋N, 1)
    using tpsG k canrepr-0 by simp
  show tps33 = tps32[70 := (⌊H⌋N, 1)]
    unfolding tps33-def tps32-def by (simp only:)
  show ttt = ttt31 + 14 + (10 + 2 * nlength 0 + 2 * nlength H)
    using assms by simp
qed

definition tps34 ≡ tpsG
  [11 := (⌊n⌋N, 1),
   15 := (⌊p n⌋N, 1),
   16 := (⌊m⌋N, 1),
```

17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6),
69 := (⌊*2*⌋$_N$, *1*),
70 := (⌊*H*⌋$_N$, *1*),
0 := (⌊*xs*⌋, *Suc n*),
69 := (⌊*2* + *2* ∗ *n*⌋$_N$, *1*)]

lemma *tm34* [*transforms-intros*]:
assumes *ttt* = *ttt31* + *24* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*)
shows *transforms tm34 tps0 ttt tps34*
unfolding *tm34-def*
proof (*tform*)
show *69* + *7* < *length tps33*
using *lentpsG tps33-def k* **by** (*simp-all only*: *length-list-update*)
let *?nss* = (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @
formula-n Φ_4 @ *formula-n* Φ_5)
show *tps33* ! *1* = *nlltape ?nss*
using *tps33-def k lentpsG* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have *tps33* ! *0* = *tps1* ! *0*
unfolding *tps33-def tpsG-def tpsF-def tpsE-def tpsD-def tpsC-def tpsB-def tpsA-def*
by (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show *tps33* ! *0* = (⌊*xs*⌋, *1*)
using *tps1'* **by** *simp*
show *tps33* ! *69* = (⌊*2*⌋$_N$, *1*)
using *tps33-def k lentpsG* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have *tps33* ! *70* = (⌊*H*⌋$_N$, *1*)
using *tps33-def k lentpsG* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show *tps33* ! (*69* + *1*) = (⌊*H*⌋$_N$, *1*)
by *simp*
have *tps33* ! *71* = *tpsG* ! *71*
using *tps33-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps33* ! (*69* + *2*) = (⌊*0*⌋$_N$, *1*)
using *tpsG k canrepr-0* **by** *simp*
have *tps33* ! *72* = *tpsG* ! *72*
using *tps33-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps33* ! (*69* + *3*) = (⌊[]⌋, *1*)
using *tpsG k* **by** *simp*
have *tps33* ! *73* = *tpsG* ! *73*
using *tps33-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps33* ! (*69* + *4*) = (⌊[]⌋, *1*)
using *tpsG k* **by** *simp*
have *tps33* ! *74* = *tpsG* ! *74*
using *tps33-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps33* ! (*69* + *5*) = (⌊[]⌋, *1*)
using *tpsG k* **by** *simp*
have *tps33* ! *75* = *tpsG* ! *75*
using *tps33-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps33* ! (*69* + *6*) = (⌊[]⌋, *1*)
using *tpsG k* **by** *simp*
have *tps33* ! *76* = *tpsG* ! *76*
using *tps33-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps33* ! (*69* + *7*) = (⌊[]⌋, *1*)
using *tpsG k* **by** *simp*
show *tps34* = *tps33*
[*0* := (⌊*xs*⌋, *Suc n*),
69 := (⌊*2* + *2* ∗ *n*⌋$_N$, *1*),

```
    1 := nlltape (?nss @ formula-n Φ6)]
   unfolding tps34-def tps33-def by (simp only: list-update-swap list-update-overwrite append-assoc)
  show ttt = ttt31 + 24 + 2 * nlength H + (133650 * H ^ 6 * n ^ 3 + 1)
   using assms by simp
qed

definition tpsH ≡ tpsG
  [69 := (⌊2⌋N, 1),
   70 := (⌊H⌋N, 1),
   0 := (⌊xs⌋, Suc n),
   69 := (⌊2 + 2 * n⌋N, 1)]

lemma tpsH: 76 < j ⟹ j < 110 ⟹ tpsH ! j = (⌊[]⌋, 1)
  using tpsH-def tpsG by simp

lemma lentpsH: length tpsH = 110
  using lentpsG tpsH-def by simp

lemma tps34: tps34 = tpsH
  [11 := (⌊n⌋N, 1),
   15 := (⌊p n⌋N, 1),
   16 := (⌊m⌋N, 1),
   17 := (⌊T'⌋N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋NL, 1),
   18 := (⌊m'⌋N, 1),
   19 := (⌊H⌋N, 1),
   20 := (⌊m' * H⌋N, 1),
   1 := nlltape
     (formula-n Φ0 @ formula-n Φ1 @ formula-n Φ2 @ formula-n Φ3 @ formula-n Φ4 @
      formula-n Φ5 @ formula-n Φ6)]
  unfolding tps34-def tpsH-def by (simp only: list-update-swap)

definition tps35 ≡ tpsH
  [11 := (⌊n⌋N, 1),
   15 := (⌊p n⌋N, 1),
   16 := (⌊m⌋N, 1),
   17 := (⌊T'⌋N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋NL, 1),
   18 := (⌊m'⌋N, 1),
   19 := (⌊H⌋N, 1),
   20 := (⌊m' * H⌋N, 1),
   1 := nlltape
     (formula-n Φ0 @ formula-n Φ1 @ formula-n Φ2 @ formula-n Φ3 @ formula-n Φ4 @
      formula-n Φ5 @ formula-n Φ6),
   77 := (⌊n⌋N, 1)]

lemma tm35 [transforms-intros]:
  assumes ttt = ttt31 + 38 + 2 * nlength H + (133650 * H ^ 6 * n ^ 3 + 1) + 3 * nlength n
  shows transforms tm35 tps0 ttt tps35
  unfolding tm35-def
proof (tform)
  show 11 < length tps34 77 < length tps34
   using lentpsH tps34 k by (simp-all only: length-list-update)
  show tps34 ! 11 = (⌊n⌋N, 1)
   using tps34 k lentpsH by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
  have tps34 ! 77 = tpsH ! 77
   using tps34 by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  then show tps34 ! 77 = (⌊0⌋N, 1)
   using tpsH k canrepr-0 by simp
  show tps35 = tps34[77 := (⌊n⌋N, 1)]
   unfolding tps35-def tps34 by (simp only: list-update-swap list-update-overwrite)
```

show *ttt* = *ttt31* + *24* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) +
(*14* + *3* ∗ (*nlength n* + *nlength 0*))
using *assms* **by** *simp*
qed

definition *tps36* ≡ *tpsH*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6),
77 := (⌊*2* ∗ *n*⌋$_N$, *1*)]

lemma *tm36* [*transforms-intros*]:
assumes *ttt* = *ttt31* + *43* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) + *5* ∗ *nlength n*
shows *transforms tm36 tps0 ttt tps36*
unfolding *tm36-def*
proof (*tform time*: *assms*)
show *77* < *length tps35*
using *lentpsH tps35-def k* **by** (*simp-all only*: *length-list-update*)
show *tps35* ! *77* = (⌊*n*⌋$_N$, *1*)
using *tps35-def k lentpsH* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show *tps36* = *tps35*[*77* := (⌊*2* ∗ *n*⌋$_N$, *1*)]
unfolding *tps36-def tps35-def* **by** (*simp only*: *list-update-swap*[*of 77*] *list-update-overwrite*)
qed

definition *tps37* ≡ *tpsH*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6),
77 := (⌊*2* ∗ *n* + *4*⌋$_N$, *1*)]

lemma *tm37* [*transforms-intros*]:
assumes *ttt* = *ttt31* + *63* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) + *5* ∗ *nlength n* +
8 ∗ *nlength* (*2* ∗ *n* + *4*)
shows *transforms tm37 tps0 ttt tps37*
unfolding *tm37-def*
proof (*tform*)
show *77* < *length tps36*
using *lentpsH tps36-def k* **by** (*simp-all only*: *length-list-update*)
show *tps36* ! *77* = (⌊*2* ∗ *n*⌋$_N$, *1*)
using *tps36-def k lentpsH* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show *tps37* = *tps36*[*77* := (⌊*2* ∗ *n* + *4*⌋$_N$, *1*)]
unfolding *tps37-def tps36-def* **by** (*simp only*: *list-update-swap list-update-overwrite*)
show *ttt* = *ttt31* + *43* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) +
5 ∗ *nlength n* + *4* ∗ (*5* + *2* ∗ *nlength* (*2* ∗ *n* + *4*))
using *assms* **by** *simp*

qed

definition *tps38* ≡ *tpsH*
[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor$*map* (λt. *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* (λt. *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6),
77 := ($\lfloor 2 * n + 4 \rfloor_N$, *1*),
78 := ($\lfloor H \rfloor_N$, *1*)]

lemma *tm38* [*transforms-intros*]:
assumes *ttt* = *ttt31* + *73* + *2* * *nlength H* + (*133650* * *H* ^ *6* * *n* ^ *3* + *1*) +
5 * *nlength n* + *8* * *nlength* (*2* * *n* + *4*) + *2* * *nlength H*
shows *transforms tm38 tps0 ttt tps38*
unfolding *tm38-def*
proof (*tform*)
show *78* < *length tps37*
using *lentpsH tps37-def k* **by** (*simp-all only*: *length-list-update*)
have *tps37* ! *78* = *tpsH* ! *78*
using *tps37-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps37* ! *78* = ($\lfloor 0 \rfloor_N$, *1*)
using *tpsH k canrepr-0* **by** *simp*
show *tps38* = *tps37*[*78* := ($\lfloor H \rfloor_N$, *1*)]
unfolding *tps38-def tps37-def* **by** (*simp only*: *list-update-swap*)
show *ttt* = *ttt31* + *63* + *2* * *nlength H* + (*133650* * *H* ^ *6* * *n* ^ *3* + *1*) +
5 * *nlength n* + *8* * *nlength* (*2* * *n* + *4*) + (*10* + *2* * *nlength 0* + *2* * *nlength H*)
using *assms* **by** *simp*
qed

definition *tps39* ≡ *tpsH*
[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor$*map* (λt. *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* (λt. *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6),
77 := ($\lfloor 2 * n + 4 \rfloor_N$, *1*),
78 := ($\lfloor H \rfloor_N$, *1*),
83 := ($\lfloor p\ n \rfloor_N$, *1*)]

lemma *tm39* [*transforms-intros*]:
assumes *ttt* = *ttt31* + *87* + *2* * *nlength H* + (*133650* * *H* ^ *6* * *n* ^ *3* + *1*) + *5* * *nlength n* +
8 * *nlength* (*2* * *n* + *4*) + *2* * *nlength H* + *3* * *nlength* (*p n*)
shows *transforms tm39 tps0 ttt tps39*
unfolding *tm39-def*
proof (*tform*)
show *15* < *length tps38 83* < *length tps38*
using *lentpsH tps38-def k* **by** (*simp-all only*: *length-list-update*)
show *tps38* ! *15* = ($\lfloor p\ n \rfloor_N$, *1*)

using *tps38-def k lentpsH* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have *tps38* ! *83* = *tpsH* ! *83*
using *tps38-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps38* ! *83* = $(\lfloor 0 \rfloor_N,\ 1)$
using *tpsH k canrepr-0* **by** *simp*
show *tps39* = *tps38*$[83 := (\lfloor p\ n \rfloor_N,\ 1)]$
unfolding *tps39-def tps38-def* **by** (*simp only*: *list-update-swap*)
show *ttt* = *ttt31* + *73* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) + *5* ∗ *nlength n* +
8 ∗ *nlength* (*2* ∗ *n* + *4*) + *2* ∗ *nlength H* + (*14* + *3* ∗ (*nlength* (*p n*) + *nlength 0*))
using *assms* **by** *simp*
qed

definition *tps40* ≡ *tpsH*
$[11 := (\lfloor n \rfloor_N,\ 1),$
$15 := (\lfloor p\ n \rfloor_N,\ 1),$
$16 := (\lfloor m \rfloor_N,\ 1),$
$17 := (\lfloor T' \rfloor_N,\ 1),$
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T'] \rfloor_{NL},\ 1),$
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T'] \rfloor_{NL},\ 1),$
$18 := (\lfloor m' \rfloor_N,\ 1),$
$19 := (\lfloor H \rfloor_N,\ 1),$
$20 := (\lfloor m' * H \rfloor_N,\ 1),$
$1 :=$ *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6 @ *formula-n* Φ_7),
$77 := (\lfloor 2 * n + 4 \rfloor_N,\ 1),$
$78 := (\lfloor H \rfloor_N,\ 1),$
$83 := (\lfloor p\ n \rfloor_N,\ 1),$
$77 := (\lfloor 2 * n + 4 + 2 * p\ n \rfloor_N,\ 1),$
$77 + 6 := (\lfloor 0 \rfloor_N,\ 1)]$

lemma *tm40* [*transforms-intros*]:
assumes *ttt* = *ttt31* + *88* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) + *5* ∗ *nlength n* +
8 ∗ *nlength* (*2* ∗ *n* + *4*) + *2* ∗ *nlength H* + *3* ∗ *nlength* (*p n*) +
p n ∗ *257* ∗ *H* ∗ $(nlength\ (2 * n + 4 + 2 * p\ n) + nlength\ H)^2$
shows *transforms tm40 tps0 ttt tps40*
unfolding *tm40-def*
proof (*tform*)
show *77* + *6* < *length tps39*
using *lentpsH tps39-def k* **by** (*simp-all only*: *length-list-update*)
let *?nss* = *formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6
show *tps39* ! *1* = *nlltape ?nss*
using *tps39-def k lentpsH* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show *tps39* ! *77* = $(\lfloor 2 * n + 4 \rfloor_N,\ 1)$
using *tps39-def k lentpsH* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have *tps39* ! *78* = $(\lfloor H \rfloor_N,\ 1)$
using *tps39-def k lentpsH* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show *tps39* ! (*77* + *1*) = $(\lfloor H \rfloor_N,\ 1)$
by *simp*
have *tps39* ! *79* = *tpsH* ! *79*
using *tps39-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps39* ! (*77* + *2*) = $(\lfloor [] \rfloor,\ 1)$
using *tpsH k* **by** *simp*
have *tps39* ! *80* = *tpsH* ! *80*
using *tps39-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps39* ! (*77* + *3*) = $(\lfloor [] \rfloor,\ 1)$
using *tpsH k* **by** *simp*
have *tps39* ! *81* = *tpsH* ! *81*
using *tps39-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps39* ! (*77* + *4*) = $(\lfloor [] \rfloor,\ 1)$
using *tpsH k* **by** *simp*
have *tps39* ! *82* = *tpsH* ! *82*

using *tps39-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show $tps39 ! (77 + 5) = (\lfloor[]\rfloor, 1)$
using *tpsH k* **by** *simp*
have $tps39 ! 83 = (\lfloor p\ n\rfloor_N, 1)$
using *tps39-def k lentpsH* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show $tps39 ! (77 + 6) = (\lfloor p\ n\rfloor_N, 1)$
by *simp*
show $tps40 = tps39$
$[77 := (\lfloor 2 * n + 4 + 2 * p\ n\rfloor_N, 1),$
$77 + 6 := (\lfloor 0\rfloor_N, 1),$
$1 := nlltape\ (?nss\ @\ formula\text{-}n\ \Phi_7)]$
unfolding *tps40-def tps39-def* **by** (*simp only*: *list-update-swap list-update-overwrite append-assoc*)
show $ttt = ttt31 + 87 + 2 * nlength\ H + (133650 * H \hat{}\ 6 * n \hat{}\ 3 + 1) + 5 * nlength\ n +$
$8 * nlength\ (2 * n + 4) + 2 * nlength\ H + 3 * nlength\ (p\ n) +$
$(p\ n * 257 * H * (nlength\ (2 * n + 4 + 2 * p\ n) + nlength\ H)^2 + 1)$
using *assms* **by** *simp*
qed

definition $tpsI \equiv tpsH$
$[77 := (\lfloor 2 * n + 4\rfloor_N, 1),$
$78 := (\lfloor H\rfloor_N, 1),$
$83 := (\lfloor p\ n\rfloor_N, 1),$
$77 := (\lfloor 2 * n + 4 + 2 * p\ n\rfloor_N, 1),$
$77 + 6 := (\lfloor 0\rfloor_N, 1)]$

lemma *tpsI*: $83 < j \Longrightarrow j < 110 \Longrightarrow tpsI ! j = (\lfloor[]\rfloor, 1)$
using *tpsI-def tpsH* **by** *simp*

lemma *lentpsI*: $length\ tpsI = 110$
using *lentpsH tpsI-def* **by** *simp*

lemma *tps40*: $tps40 = tpsI$
$[11 := (\lfloor n\rfloor_N, 1),$
$15 := (\lfloor p\ n\rfloor_N, 1),$
$16 := (\lfloor m\rfloor_N, 1),$
$17 := (\lfloor T'\rfloor_N, 1),$
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T']\rfloor_{NL}, 1),$
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T']\rfloor_{NL}, 1),$
$18 := (\lfloor m'\rfloor_N, 1),$
$19 := (\lfloor H\rfloor_N, 1),$
$20 := (\lfloor m' * H\rfloor_N, 1),$
$1 := nlltape$
$(formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1\ @\ formula\text{-}n\ \Phi_2\ @\ formula\text{-}n\ \Phi_3\ @\ formula\text{-}n\ \Phi_4\ @$
$formula\text{-}n\ \Phi_5\ @\ formula\text{-}n\ \Phi_6\ @\ formula\text{-}n\ \Phi_7)]$
unfolding *tps40-def tpsI-def* **by** (*simp only*: *list-update-swap*)

definition $tps41 \equiv tpsI$
$[11 := (\lfloor n\rfloor_N, 1),$
$15 := (\lfloor p\ n\rfloor_N, 1),$
$16 := (\lfloor m\rfloor_N, 1),$
$17 := (\lfloor T'\rfloor_N, 1),$
$4 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 0)\ [0..<Suc\ T']\rfloor_{NL}, 1),$
$7 := (\lfloor map\ (\lambda t.\ exc\ zs\ t <\#> 1)\ [0..<Suc\ T']\rfloor_{NL}, 1),$
$18 := (\lfloor m'\rfloor_N, 1),$
$19 := (\lfloor H\rfloor_N, 1),$
$20 := (\lfloor m' * H\rfloor_N, 1),$
$1 := nlltape$
$(formula\text{-}n\ \Phi_0\ @\ formula\text{-}n\ \Phi_1\ @\ formula\text{-}n\ \Phi_2\ @\ formula\text{-}n\ \Phi_3\ @\ formula\text{-}n\ \Phi_4\ @$
$formula\text{-}n\ \Phi_5\ @\ formula\text{-}n\ \Phi_6\ @\ formula\text{-}n\ \Phi_7),$
$84 := (\lfloor m'\rfloor_N, 1)]$

lemma *tm41* [*transforms-intros*]:
assumes $ttt = ttt31 + 102 + 2 * nlength\ H + (133650 * H \hat{}\ 6 * n \hat{}\ 3 + 1) + 5 * nlength\ n +$

8 ∗ nlength (2 ∗ n + 4) + 2 ∗ nlength H + 3 ∗ nlength (p n) +
p n ∗ 257 ∗ H ∗ (nlength (2 ∗ n + 4 + 2 ∗ p n) + nlength H)^2 *+*
3 ∗ nlength m′
shows *transforms tm41 tps0 ttt tps41*
unfolding *tm41-def*
proof (*tform*)
show *18 < length tps40 84 < length tps40*
using *lentpsI tps40 k* **by** (*simp-all only*: *length-list-update*)
show *tps40 ! 18 =* ($\lfloor m' \rfloor_N$, *1*)
using *tps40-def k lentpsH* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have *tps40 ! 84 = tpsI ! 84*
using *tps40* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps40 ! 84 =* ($\lfloor 0 \rfloor_N$, *1*)
using *tpsI k canrepr-0* **by** *simp*
show *tps41 = tps40*[*84 :=* ($\lfloor m' \rfloor_N$, *1*)]
unfolding *tps41-def tps40* **by** (*simp only*: *list-update-swap*)
show *ttt = ttt31 + 88 + 2 ∗ nlength H + (133650 ∗ H ^ 6 ∗ n ^ 3 + 1) + 5 ∗ nlength n +*
8 ∗ nlength (2 ∗ n + 4) + 2 ∗ nlength H + 3 ∗ nlength (p n) +
p n ∗ 257 ∗ H ∗ (nlength (2 ∗ n + 4 + 2 ∗ p n) + nlength H)^2 *+*
(14 + 3 ∗ (nlength m′ + nlength 0))
using *assms* **by** *simp*
qed

definition *tps42 ≡ tpsI*
[*11 :=* ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor$*map (λt. exc zs t <#> 0) [0..<Suc T′]*$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map (λt. exc zs t <#> 1) [0..<Suc T′]*$\rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),
1 := nlltape
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6 @ *formula-n* Φ_7),
84 := ($\lfloor T' + m' \rfloor_N$, *1*)]

lemma *tm42* [*transforms-intros*]:
assumes *ttt = ttt31 + 112 + 2 ∗ nlength H + (133650 ∗ H ^ 6 ∗ n ^ 3 + 1) + 5 ∗ nlength n +*
8 ∗ nlength (2 ∗ n + 4) + 2 ∗ nlength H + 3 ∗ nlength (p n) +
p n ∗ 257 ∗ H ∗ (nlength (2 ∗ n + 4 + 2 ∗ p n) + nlength H)^2 *+ 3 ∗ nlength m′ +*
3 ∗ max (nlength T′) (nlength m′)
shows *transforms tm42 tps0 ttt tps42*
unfolding *tm42-def*
proof (*tform*)
show *17 < length tps41 84 < length tps41*
using *lentpsI tps41-def k* **by** (*simp-all only*: *length-list-update*)
show *tps41 ! 17 =* ($\lfloor T' \rfloor_N$, *1*)
using *tps41-def k lentpsI* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show *tps41 ! 84 =* ($\lfloor m' \rfloor_N$, *1*)
using *tps41-def k lentpsI* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show *tps42 = tps41*[*84 :=* ($\lfloor T' + m' \rfloor_N$, *1*)]
unfolding *tps42-def tps41-def* **by** (*simp only*: *list-update-overwrite*)
show *ttt = ttt31 + 102 + 2 ∗ nlength H + (133650 ∗ H ^ 6 ∗ n ^ 3 + 1) + 5 ∗ nlength n +*
8 ∗ nlength (2 ∗ n + 4) + 2 ∗ nlength H + 3 ∗ nlength (p n) +
p n ∗ 257 ∗ H ∗ (nlength (2 ∗ n + 4 + 2 ∗ p n) + nlength H)^2 *+*
3 ∗ nlength m′ + (3 ∗ max (nlength T′) (nlength m′) + 10)
using *assms* **by** *simp*
qed

definition *tps43 ≡ tpsI*
[*11 :=* ($\lfloor n \rfloor_N$, *1*),

15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6 @ *formula-n* Φ_7),
84 := (⌊*2* ∗ *T'* + *m'*⌋$_N$, *1*)]

lemma *tm43* [*transforms-intros*]:
assumes *ttt* = *ttt31* + *122* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) + *5* ∗ *nlength n* +
8 ∗ *nlength* (*2* ∗ *n* + *4*) + *2* ∗ *nlength H* + *3* ∗ *nlength* (*p n*) +
p n ∗ *257* ∗ *H* ∗ (*nlength* (*2* ∗ *n* + *4* + *2* ∗ *p n*) + *nlength H*)^2 + *3* ∗ *nlength m'* +
3 ∗ *max* (*nlength T'*) (*nlength m'*) +
3 ∗ *max* (*nlength T'*) (*nlength* (*T'* + *m'*))
shows *transforms tm43 tps0 ttt tps43*
unfolding *tm43-def*
proof (*tform*)
show *17* < *length tps42 84* < *length tps42*
using *lentpsI tps42-def k* **by** (*simp-all only*: *length-list-update*)
show *tps42* ! *17* = (⌊*T'*⌋$_N$, *1*)
using *tps42-def k lentpsI* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show *tps42* ! *84* = (⌊*T'* + *m'*⌋$_N$, *1*)
using *tps42-def k lentpsI* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have *tps43* = *tps42*[*84* := (⌊*2* ∗ *T'* + *m'*⌋$_N$, *1*)]
unfolding *tps43-def tps42-def* **by** (*simp only*: *list-update-overwrite*)
then show *tps43* = *tps42*[*84* := (⌊*T'* + (*T'* + *m'*)⌋$_N$, *1*)]
by (*simp add*: *left-add-twice*)
show *ttt* = *ttt31* + *112* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) + *5* ∗ *nlength n* +
8 ∗ *nlength* (*2* ∗ *n* + *4*) + *2* ∗ *nlength H* + *3* ∗ *nlength* (*p n*) +
p n ∗ *257* ∗ *H* ∗ (*nlength* (*2* ∗ *n* + *4* + *2* ∗ *p n*) + *nlength H*)^2 + *3* ∗ *nlength m'* +
3 ∗ *max* (*nlength T'*) (*nlength m'*) +
(*3* ∗ *max* (*nlength T'*) (*nlength* (*T'* + *m'*)) + *10*)
using *assms* **by** *simp*
qed

definition *tps44* ≡ *tpsI*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T'*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]⌋$_{NL}$, *1*),
18 := (⌊*m'*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m'* ∗ *H*⌋$_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6 @ *formula-n* Φ_7),
84 := (⌊*3* ∗ *T'* + *m'*⌋$_N$, *1*)]

lemma *tm44* [*transforms-intros*]:
assumes *ttt* = *ttt31* + *132* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) + *5* ∗ *nlength n* +
8 ∗ *nlength* (*2* ∗ *n* + *4*) + *2* ∗ *nlength H* + *3* ∗ *nlength* (*p n*) +
p n ∗ *257* ∗ *H* ∗ (*nlength* (*2* ∗ *n* + *4* + *2* ∗ *p n*) + *nlength H*)^2 + *3* ∗ *nlength m'* +
3 ∗ *max* (*nlength T'*) (*nlength m'*) +
3 ∗ *max* (*nlength T'*) (*nlength* (*T'* + *m'*)) +
3 ∗ *max* (*nlength T'*) (*nlength* (*2* ∗ *T'* + *m'*))
shows *transforms tm44 tps0 ttt tps44*

unfolding *tm44-def*
proof (*tform*)
show *17 < length tps43 84 < length tps43*
using *lentpsI tps43-def k* **by** (*simp-all only*: *length-list-update*)
show *tps43 ! 17* = ($\lfloor T' \rfloor_N$, *1*)
using *tps43-def k lentpsI* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show *tps43 ! 84* = ($\lfloor 2 * T' + m' \rfloor_N$, *1*)
using *tps43-def k lentpsI* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have *tps44* = *tps43*[*84* := ($\lfloor 3 * T' + m' \rfloor_N$, *1*)]
unfolding *tps44-def tps43-def* **by** (*simp only*: *list-update-overwrite*)
then show *tps44* = *tps43*[*84* := ($\lfloor T' + (2 * T' + m') \rfloor_N$, *1*)]
by (*simp add*: *left-add-twice*)
show $ttt = ttt31 + 122 + 2 * nlength\ H + (133650 * H \hat{}\ 6 * n \hat{}\ 3 + 1) + 5 * nlength\ n +$
$8 * nlength\ (2 * n + 4) + 2 * nlength\ H + 3 * nlength\ (p\ n) +$
$p\ n * 257 * H * (nlength\ (2 * n + 4 + 2 * p\ n) + nlength\ H)^2 + 3 * nlength\ m' +$
$3 * max\ (nlength\ T')\ (nlength\ m') +$
$3 * max\ (nlength\ T')\ (nlength\ (T' + m')) +$
$(3 * max\ (nlength\ T')\ (nlength\ (2 * T' + m')) + 10)$
using *assms* **by** *simp*
qed

definition *tps45* ≡ *tpsI*
[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor map\ (\lambda t.\ exc\ zs\ t$ <#> $0)\ [0..<Suc\ T'] \rfloor_{NL}$, *1*),
7 := ($\lfloor map\ (\lambda t.\ exc\ zs\ t$ <#> $1)\ [0..<Suc\ T'] \rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6 @ *formula-n* Φ_7),
84 := ($\lfloor 1 + 3 * T' + m' \rfloor_N$, *1*)]

lemma *tm45* [*transforms-intros*]:
assumes $ttt = ttt31 + 137 + 2 * nlength\ H + (133650 * H \hat{}\ 6 * n \hat{}\ 3 + 1) + 5 * nlength\ n +$
$8 * nlength\ (2 * n + 4) + 2 * nlength\ H + 3 * nlength\ (p\ n) +$
$p\ n * 257 * H * (nlength\ (2 * n + 4 + 2 * p\ n) + nlength\ H)^2 + 3 * nlength\ m' +$
$3 * max\ (nlength\ T')\ (nlength\ m') +$
$3 * max\ (nlength\ T')\ (nlength\ (T' + m')) +$
$3 * max\ (nlength\ T')\ (nlength\ (2 * T' + m')) +$
$2 * nlength\ (3 * T' + m')$
shows *transforms tm45 tps0 ttt tps45*
unfolding *tm45-def*
proof (*tform*)
show *84 < length tps44*
using *lentpsI tps44-def k* **by** (*simp-all only*: *length-list-update*)
show *tps44 ! 84* = ($\lfloor 3 * T' + m' \rfloor_N$, *1*)
using *tps44-def k lentpsI* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have *tps45* = *tps44*[*84* := ($\lfloor 1 + 3 * T' + m' \rfloor_N$, *1*)]
unfolding *tps45-def tps44-def* **by** (*simp only*: *list-update-overwrite*)
then show *tps45* = *tps44*[*84* := ($\lfloor Suc\ (3 * T' + m') \rfloor_N$, *1*)]
by *simp*
show $ttt = ttt31 + 132 + 2 * nlength\ H + (133650 * H \hat{}\ 6 * n \hat{}\ 3 + 1) + 5 * nlength\ n +$
$8 * nlength\ (2 * n + 4) + 2 * nlength\ H + 3 * nlength\ (p\ n) +$
$p\ n * 257 * H * (nlength\ (2 * n + 4 + 2 * p\ n) + nlength\ H)^2 + 3 * nlength\ m' +$
$3 * max\ (nlength\ T')\ (nlength\ m') +$
$3 * max\ (nlength\ T')\ (nlength\ (T' + m')) +$
$3 * max\ (nlength\ T')\ (nlength\ (2 * T' + m')) +$
$(5 + 2 * nlength\ (3 * T' + m'))$
using *assms* **by** *simp*

qed

definition *tps46* ≡ *tpsI*
[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6 @ *formula-n* Φ_7),
84 := ($\lfloor 1 + 3 * T' + m' \rfloor_N$, *1*),
85 := ($\lfloor H \rfloor_N$, *1*)]

lemma *tm46* [*transforms-intros*]:
assumes *ttt* = *ttt31* + *147* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) + *5* ∗ *nlength n* +
8 ∗ *nlength* (*2* ∗ *n* + *4*) + *4* ∗ *nlength H* + *3* ∗ *nlength* (*p n*) +
p n ∗ *257* ∗ *H* ∗ (*nlength* (*2* ∗ *n* + *4* + *2* ∗ *p n*) + *nlength H*)^2 + *3* ∗ *nlength m'* +
3 ∗ *max* (*nlength T'*) (*nlength m'*) +
3 ∗ *max* (*nlength T'*) (*nlength* (*T'* + *m'*)) +
3 ∗ *max* (*nlength T'*) (*nlength* (*2* ∗ *T'* + *m'*)) +
2 ∗ *nlength* (*3* ∗ *T'* + *m'*)
shows *transforms tm46 tps0 ttt tps46*
unfolding *tm46-def*
proof (*tform*)
show *85* < *length tps45*
using *lentpsI tps45-def k* **by** (*simp-all only*: *length-list-update*)
have *tps45* ! *85* = *tpsI* ! *85*
using *tps45-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps45* ! *85* = ($\lfloor 0 \rfloor_N$, *1*)
using *tpsI k canrepr-0* **by** *simp*
show *tps46* = *tps45*[*85* := ($\lfloor H \rfloor_N$, *1*)]
unfolding *tps46-def tps45-def* **by** (*simp only*:)
show *ttt* = *ttt31* + *137* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) + *5* ∗ *nlength n* +
8 ∗ *nlength* (*2* ∗ *n* + *4*) + *2* ∗ *nlength H* + *3* ∗ *nlength* (*p n*) +
p n ∗ *257* ∗ *H* ∗ (*nlength* (*2* ∗ *n* + *4* + *2* ∗ *p n*) + *nlength H*)^2 + *3* ∗ *nlength m'* +
3 ∗ *max* (*nlength T'*) (*nlength m'*) +
3 ∗ *max* (*nlength T'*) (*nlength* (*T'* + *m'*)) +
3 ∗ *max* (*nlength T'*) (*nlength* (*2* ∗ *T'* + *m'*)) +
2 ∗ *nlength* (*3* ∗ *T'* + *m'*) +
(*10* + *2* ∗ *nlength 0* + *2* ∗ *nlength H*)
using *assms* **by** *simp*
qed

definition *tps47* ≡ *tpsI*
[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),
17 := ($\lfloor T' \rfloor_N$, *1*),
4 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m' \rfloor_N$, *1*),
19 := ($\lfloor H \rfloor_N$, *1*),
20 := ($\lfloor m' * H \rfloor_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6 @ *formula-n* Φ_7 @ *formula-n* Φ_8),
84 := ($\lfloor 1 + 3 * T' + m' \rfloor_N$, *1*),
85 := ($\lfloor H \rfloor_N$, *1*),

84 + *2* := ($\lfloor 3 \rfloor_N$, *1*),
84 + *6* := ($\lfloor$*formula-n* $\Phi_8 \rfloor_{NLL}$, *1*)]

definition *ttt47* ≡ *ttt31* + *166* +
6 ∗ *nlength H* +
133650 ∗ *H* ^ *6* ∗ *n* ^ *3* +
5 ∗ *nlength n* +
8 ∗ *nlength* (*2* ∗ *n* + *4*) +
3 ∗ *nlength* (*p n*) +
p n ∗ *257* ∗ *H* ∗ (*nlength* (*2* ∗ *n* + *4* + *2* ∗ *p n*) + *nlength H*)^2 +
3 ∗ *nlength m'* +
3 ∗ *max* (*nlength T'*) (*nlength m'*) +
3 ∗ *max* (*nlength T'*) (*nlength* (*T'* + *m'*)) +
3 ∗ *max* (*nlength T'*) (*nlength* (*2* ∗ *T'* + *m'*)) +
2 ∗ *nlength* (*3* ∗ *T'* + *m'*) +
1861 ∗ *H* ^ *4* ∗ (*nlength* (*Suc* (*1* + *3* ∗ *T'* + *m'*)))^2

lemma *ttt47*: *ttt47* ≤ (*32* ∗ *d-polynomial p* + *364343*) ∗ *H* ^ *6* ∗ *N* ^ *4*
proof −
have *nlength-le-GN*: *y* ≤ *N* ⟹ *nlength y* ≤ *H* ^ *6* ∗ *N* ^ *3* **for** *y*
proof −
assume *y* ≤ *N*
then have *nlength y* ≤ *N* ^ *3*
using *nlength-le-n linear-le-pow H-ge-3* **by** (*meson dual-order.trans zero-less-numeral*)
also have ... ≤ *H* ^ *6* ∗ *N* ^ *3*
using *H-gr-2* **by** *simp*
finally show *?thesis* **.**
qed

have *h*: *6* ∗ *nlength H* ≤ *6* ∗ *H* ^ *6* ∗ *N* ^ *3*
using *nlength-le-GN*[*OF H-le-N*] **by** *simp*
have *i*: *5* ∗ *nlength n* ≤ *5* ∗ *H* ^ *6* ∗ *N* ^ *3*
using *nlength-le-GN*[*OF n-le-N*] **by** *simp*
have *j*: *8* ∗ *nlength* (*2* ∗ *n* + *4*) ≤ *8* ∗ *H* ^ *6* ∗ *N* ^ *3*
proof −
have *2* ∗ *n* + *4* ≤ *2* ∗ *n* + *H* ∗ *3*
using *H-ge-3* **by** *simp*
also have ... ≤ *H* ∗ *2* ∗ *n* + *H* ∗ *3*
using *H-ge-3* **by** *simp*
also have ... = *H* ∗ (*2* ∗ *n* + *3*)
by *algebra*
also have ... ≤ *N*
using *N-eq* **by** *simp*
finally have *2* ∗ *n* + *4* ≤ *N* **.**
then have *8* ∗ *nlength* (*2* ∗ *n* + *4*) ≤ *8* ∗ *nlength N*
using *nlength-mono* **by** *simp*
also have ... ≤ *8* ∗ *N*
using *nlength-le-n* **by** *simp*
also have ... ≤ *8* ∗ *H* ^ *6* ∗ *N*
using *H-ge-3* **by** *simp*
also have ... ≤ *8* ∗ *H* ^ *6* ∗ *N* ^ *3*
using *linear-le-pow* **by** *simp*
finally show *?thesis* **.**
qed
have *k*: *3* ∗ *nlength* (*p n*) ≤ *3* ∗ *H* ^ *6* ∗ *N* ^ *3*
using *nlength-le-GN*[*OF le-N*] **by** *simp*
have *l*: *3* ∗ *nlength m'* ≤ *3* ∗ *H* ^ *6* ∗ *N* ^ *3*
using *nlength-le-GN*[*OF le-N*] *m'-def* **by** *simp*
have *g*: *3* ∗ *max* (*nlength T'*) (*nlength m'*) ≤ *3* ∗ *H* ^ *6* ∗ *N* ^ *3*
proof −
have *m'* ≤ *N*
using *le-N m'-def* **by** *simp*
moreover have *T'* ≤ *N*

using *le-N* **by** *simp*
ultimately have $max\ (nlength\ T')\ (nlength\ m') \le nlength\ N$
using *max-nlength nlength-mono* **by** *simp*
then have $3 * max\ (nlength\ T')\ (nlength\ m') \le 3 * N$
using *nlength-le-n* **by** (*meson le-trans mult-le-mono2*)
also have ... $\le 3 * H\ \hat{}\ 6 * N$
using *H-ge-3* **by** *simp*
also have ... $\le 3 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
using *linear-le-pow* **by** *simp*
finally show *?thesis* **.**
qed
have *f*: $3 * max\ (nlength\ T')\ (nlength\ (T' + m')) \le 6 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
proof −
have $T' + m' \le N + N$
using *N-eq m′-def H-gr-2 add-le-mono le-N less-or-eq-imp-le mult-le-mono trans-le-add2*
by *presburger*
then have $T' + m' \le 2 * N$
by *simp*
moreover have $T' \le N$
using *le-N* **by** *simp*
ultimately have $max\ (nlength\ T')\ (nlength\ (T' + m')) \le nlength\ (2 * N)$
using *max-nlength nlength-mono* **by** *simp*
then have $3 * max\ (nlength\ T')\ (nlength\ (T' + m')) \le 3 * (2 * N)$
using *nlength-le-n* **by** (*meson le-trans mult-le-mono2*)
also have ... $= 6 * N$
by *simp*
also have ... $\le 6 * H\ \hat{}\ 6 * N$
using *H-ge-3* **by** *simp*
also have ... $\le 6 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
using *linear-le-pow* **by** *simp*
finally show *?thesis* **.**
qed
have *e*: $3 * max\ (nlength\ T')\ (nlength\ (2 * T' + m')) \le 6 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
proof −
have $2 * T' + m' \le N + N$
using *N-eq m′-def H-gr-2 add-le-mono le-N less-or-eq-imp-le mult-le-mono trans-le-add2*
by *presburger*
then have $2 * T' + m' \le 2 * N$
by *simp*
moreover have $T' \le N$
using *le-N* **by** *simp*
ultimately have $max\ (nlength\ T')\ (nlength\ (2 * T' + m')) \le nlength\ (2 * N)$
using *max-nlength nlength-mono* **by** *simp*
then have $3 * max\ (nlength\ T')\ (nlength\ (2 * T' + m')) \le 3 * (2 * N)$
using *nlength-le-n* **by** (*meson le-trans mult-le-mono2*)
also have ... $= 6 * N$
by *simp*
also have ... $\le 6 * H\ \hat{}\ 6 * N$
using *H-ge-3* **by** *simp*
also have ... $\le 6 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
using *linear-le-pow* **by** *simp*
finally show *?thesis* **.**
qed
have *d*: $2 * nlength\ (3 * T' + m') \le 4 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
proof −
have $3 * T' + m' \le N + N$
using *N-eq H-ge-3 m′-def* **by** (*metis add-leE add-le-mono le-N le-refl mult-le-mono*)
then have $3 * T' + m' \le 2 * N$
by *simp*
then have $nlength\ (3 * T' + m') \le 2 * N$
using *nlength-le-n le-trans* **by** *blast*
then have $2 * nlength\ (3 * T' + m') \le 4 * N$
by *simp*

also have ... $\leq 4 * H\ \hat{}\ 6 * N$
using *H-ge-3* **by** *simp*
also have ... $\leq 4 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
using *linear-le-pow* **by** *simp*
finally show *?thesis* **.**
qed
have *c*: $6 * nlength\ H \leq 6 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
proof −
have $6 * nlength\ H \leq 6 * H$
using *nlength-le-n* **by** *simp*
also have ... $\leq 6 * H\ \hat{}\ 6$
using *linear-le-pow* **by** *simp*
also have ... $\leq 6 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
using *N-ge-1* **by** *simp*
finally show *?thesis* **.**
qed
have *a*: $p\ n * 257 * H * (nlength\ (2 * n + 4 + 2 * p\ n) + nlength\ H)^2 \leq 1028 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
proof −
have $nlength\ (2 * n + 4 + 2 * p\ n) = nlength\ (2 * (n + 2 + p\ n))$
by (*metis distrib-left-numeral mult-2-right numeral-Bit0*)
also have ... $\leq Suc\ (nlength\ (n + 2 + p\ n))$
using *nlength-times2* **by** *blast*
also have ... $\leq Suc\ (n + 2 + p\ n)$
by (*simp add*: *nlength-le-n*)
also have ... $\leq N$
using *le-N* **by** *simp*
finally have $nlength\ (2 * n + 4 + 2 * p\ n) \leq N$ **.**
then have $(nlength\ (2 * n + 4 + 2 * p\ n) + nlength\ H)^2 \leq (N + nlength\ H)\ \hat{}\ 2$
by *simp*
also have ... $\leq (N + N)\ \hat{}\ 2$
using *H-le-N nlength-le-n* **by** (*meson add-left-mono le-trans power2-nat-le-eq-le*)
also have ... $= (2 * N)\ \hat{}\ 2$
by *algebra*
also have ... $= 4 * N\ \hat{}\ 2$
by *simp*
finally have $(nlength\ (2 * n + 4 + 2 * p\ n) + nlength\ H)^2 \leq 4 * N\ \hat{}\ 2$ **.**
then have $p\ n * 257 * H * (nlength\ (2 * n + 4 + 2 * p\ n) + nlength\ H)^2 \leq p\ n * 257 * H * (4 * N\ \hat{}\ 2)$
by *simp*
also have ... $\leq N * 257 * H * (4 * N\ \hat{}\ 2)$
using *le-N* **by** *simp*
also have ... $= 1028 * H * N\ \hat{}\ 3$
by *algebra*
also have ... $\leq 1028 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
using *linear-le-pow* **by** *simp*
finally show *?thesis* **.**
qed
have *b*: $1861 * H\ \hat{}\ 4 * (nlength\ (Suc\ (1 + 3 * T' + m')))^2 \leq 7444 * H\ \hat{}\ 6 * N\ \hat{}\ 3$
proof −
have $Suc\ (1 + 3 * T' + m') \leq 3 * (2 * n + 2 * p\ n + 3 + T') + m'$
by *simp*
also have ... $\leq N + N$
using *N-eq H-ge-3 m′-def add-le-mono le-N le-refl mult-le-mono1* **by** *presburger*
also have ... $\leq 2 * N$
by *simp*
finally have $Suc\ (1 + 3 * T' + m') \leq 2 * N$ **.**
then have $nlength\ (Suc\ (1 + 3 * T' + m')) \leq 2 * N$
using *nlength-le-n le-trans* **by** *blast*
then have $(nlength\ (Suc\ (1 + 3 * T' + m')))\ \hat{}\ 2 \leq (2 * N)\ \hat{}\ 2$
using *power2-nat-le-eq-le* **by** *presburger*
then have $(nlength\ (Suc\ (1 + 3 * T' + m')))\ \hat{}\ 2 \leq 4 * N\ \hat{}\ 2$
by *simp*
then have $1861 * H\ \hat{}\ 4 * (nlength\ (Suc\ (1 + 3 * T' + m')))^2 \leq 1861 * H\ \hat{}\ 4 * (4 * N\ \hat{}\ 2)$
by *simp*

```
    also have ... = 7444 * H ^ 4 * N ^ 2
      by simp
    also have ... ≤ 7444 * H ^ 6 * N ^ 2
      using pow-mono' by simp
    also have ... ≤ 7444 * H ^ 6 * N ^ 3
      using pow-mono' by simp
    finally show ?thesis .
  qed
  have m: 133650 * H ^ 6 * n ^ 3 ≤ 133650 * H ^ 6 * N ^ 3
    using n-le-N by simp

  have ttt47 ≤ ttt31 + 166 +
      6 * H ^ 6 * N ^ 3 + 133650 * H ^ 6 * N ^ 3 + 5 * H ^ 6 * N ^ 3 +
      8 * H ^ 6 * N ^ 3 + 3 * H ^ 6 * N ^ 3 + 1028 * H ^ 6 * N ^ 3 +
      3 * H ^ 6 * N ^ 3 + 3 * H ^ 6 * N ^ 3 + 6 * H ^ 6 * N ^ 3 +
      6 * H ^ 6 * N ^ 3 + 4 * H ^ 6 * N ^ 3 + 7444 * H ^ 6 * N ^ 3
    using ttt47-def a b c d e f g h i j k l m by linarith
  also have ... = ttt31 + 166 + 142166 * H ^ 6 * N ^ 3
    by simp
  also have ... ≤ ttt31 + 166 * H ^ 6 + 142166 * H ^ 6 * N ^ 3
    using H-ge-3 by simp
  also have ... ≤ ttt31 + 166 * H ^ 6 * N ^ 3 + 142166 * H ^ 6 * N ^ 3
    using N-ge-1 by simp
  also have ... = ttt31 + 142332 * H ^ 6 * N ^ 3
    by simp
  also have ... ≤ ttt31 + 142332 * H ^ 6 * N ^ 4
    using pow-mono' by simp
  also have ... ≤ (32 * d-polynomial p + 222011) * H ^ 4 * N ^ 4 + 142332 * H ^ 6 * N ^ 4
    using ttt31 by simp
  also have ... ≤ (32 * d-polynomial p + 222011) * H ^ 6 * N ^ 4 + 142332 * H ^ 6 * N ^ 4
    using pow-mono' by simp
  also have ... = (32 * d-polynomial p + 364343) * H ^ 6 * N ^ 4
    by algebra
  finally show ?thesis .
qed

lemma tm47 [transforms-intros]: transforms tm47 tps0 ttt47 tps47
  unfolding tm47-def
proof (tform)
  show 84 + 7 < length tps46
    using lentpsI tps46-def k by (simp-all only: length-list-update)
  let ?nss = formula-n Φ_0 @ formula-n Φ_1 @ formula-n Φ_2 @ formula-n Φ_3 @ formula-n Φ_4 @
      formula-n Φ_5 @ formula-n Φ_6 @ formula-n Φ_7
  show tps46 ! 1 = nlltape ?nss
    using tps46-def k lentpsI by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
  show tps46 ! 84 = (⌊1 + 3 * T' + m'⌋_N, 1)
    using tps46-def k lentpsI by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
  have tps46 ! 85 = (⌊H⌋_N, 1)
    using tps46-def k lentpsI by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
  then show tps46 ! (84 + 1) = (⌊H⌋_N, 1)
    by simp
  have tps46 ! 86 = tpsI ! 86
    using tps46-def by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  then show tps46 ! (84 + 2) = (⌊[]⌋, 1)
    using tpsI k by simp
  have tps46 ! 87 = tpsI ! 87
    using tps46-def by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  then show tps46 ! (84 + 3) = (⌊[]⌋, 1)
    using tpsI k by simp
  have tps46 ! 88 = tpsI ! 88
    using tps46-def by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  then show tps46 ! (84 + 4) = (⌊[]⌋, 1)
    using tpsI k by simp
```

have *tps46* ! *89* = *tpsI* ! *89*
  using *tps46-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps46* ! (*84* + *5*) = (⌊[]⌋, *1*)
  using *tpsI k* **by** *simp*
have *tps46* ! *90* = *tpsI* ! *90*
  using *tps46-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps46* ! (*84* + *6*) = (⌊[]⌋, *1*)
  using *tpsI k* **by** *simp*
have *tps46* ! *91* = *tpsI* ! *91*
  using *tps46-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps46* ! (*84* + *7*) = (⌊[]⌋, *1*)
  using *tpsI k* **by** *simp*
show *tps47* = *tps46*
    [*1* := *nlltape* (*?nss* @ *formula-n* Φ_8),
     84 + *2* := (⌊*3*⌋$_N$, *1*),
     84 + *6* := (⌊*formula-n* Φ_8⌋$_{NLL}$, *1*)]
  unfolding *tps47-def tps46-def* **by** (*simp only*: *list-update-swap list-update-overwrite append-assoc*)
show *ttt47* = *ttt31* + *147* + *2* ∗ *nlength H* + (*133650* ∗ *H* ^ *6* ∗ *n* ^ *3* + *1*) + *5* ∗ *nlength n* +
    8 ∗ *nlength* (*2* ∗ *n* + *4*) + *4* ∗ *nlength H* + *3* ∗ *nlength* (*p n*) +
    p n ∗ *257* ∗ *H* ∗ (*nlength* (*2* ∗ *n* + *4* + *2* ∗ *p n*) + *nlength H*)^2 + *3* ∗ *nlength m′* +
    3 ∗ *max* (*nlength T′*) (*nlength m′*) +
    3 ∗ *max* (*nlength T′*) (*nlength* (*T′* + *m′*)) +
    3 ∗ *max* (*nlength T′*) (*nlength* (*2* ∗ *T′* + *m′*)) +
    2 ∗ *nlength* (*3* ∗ *T′* + *m′*) +
    (*18* + *1861* ∗ *H* ^ *4* ∗ (*nlength* (*Suc* (*1* + *3* ∗ *T′* + *m′*)))^2)
  using *ttt47-def* **by** *simp*
qed

definition *tpsJ* ≡ *tpsI*
  [*84* := (⌊*1* + *3* ∗ *T′* + *m′*⌋$_N$, *1*),
   85 := (⌊*H*⌋$_N$, *1*),
   84 + *2* := (⌊*3*⌋$_N$, *1*),
   84 + *6* := (⌊*formula-n* Φ_8⌋$_{NLL}$, *1*)]

lemma *tpsJ*: *90* < *j* ⟹ *j* < *110* ⟹ *tpsJ* ! *j* = (⌊[]⌋, *1*)
  using *tpsJ-def tpsI* **by** *simp*

lemma *lentpsJ*: *length tpsJ* = *110*
  using *lentpsI tpsJ-def* **by** *simp*

lemma *tps47*: *tps47* = *tpsJ*
  [*11* := (⌊*n*⌋$_N$, *1*),
   15 := (⌊*p n*⌋$_N$, *1*),
   16 := (⌊*m*⌋$_N$, *1*),
   17 := (⌊*T′*⌋$_N$, *1*),
   4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
   7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
   18 := (⌊*m′*⌋$_N$, *1*),
   19 := (⌊*H*⌋$_N$, *1*),
   20 := (⌊*m′* ∗ *H*⌋$_N$, *1*),
   1 := *nlltape*
     (*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
      formula-n Φ_5 @ *formula-n* Φ_6 @ *formula-n* Φ_7 @ *formula-n* Φ_8)]
  unfolding *tps47-def tpsJ-def* **by** (*simp only*: *list-update-swap*)

definition *tps48* ≡ *tpsJ*
  [*11* := (⌊*n*⌋$_N$, *1*),
   15 := (⌊*p n*⌋$_N$, *1*),
   16 := (⌊*m*⌋$_N$, *1*),
   17 := (⌊*T′*⌋$_N$, *1*),
   4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
   7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
   18 := (⌊*m′*⌋$_N$, *1*),

```
    19 := (⌊H⌋N, 1),
    20 := (⌊m' * H⌋N, 1),
    1 := nlltape
      (formula-n Φ0 @ formula-n Φ1 @ formula-n Φ2 @ formula-n Φ3 @ formula-n Φ4 @
       formula-n Φ5 @ formula-n Φ6 @ formula-n Φ7 @ formula-n Φ8),
    91 := (⌊N⌋N, 1)]

lemma tm48 [transforms-intros]:
  assumes ttt = ttt47 + 14 + 3 * nlength N
  shows transforms tm48 tps0 ttt tps48
  unfolding tm48-def
proof (tform)
  show 20 < length tps47 91 < length tps47
    using lentpsJ tps47 k by (simp-all only: length-list-update)
  have tps47 ! 20 = (⌊m' * H⌋N, 1)
    using tps47 k lentpsJ by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
  then show tps47 ! 20 = (⌊N⌋N, 1)
    using m' by simp
  have tps47 ! 91 = tpsJ ! 91
    using tps47 by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  then show tps47 ! 91 = (⌊0⌋N, 1)
    using tpsJ k canrepr-0 by simp
  show tps48 = tps47[91 := (⌊N⌋N, 1)]
    unfolding tps48-def tps47 by (simp only: list-update-swap list-update-overwrite)
  show ttt = ttt47 + (14 + 3 * (nlength N + nlength 0))
    using assms by simp
qed

definition tps49 ≡ tpsJ
  [11 := (⌊n⌋N, 1),
    15 := (⌊p n⌋N, 1),
    16 := (⌊m⌋N, 1),
    17 := (⌊T'⌋N, 1),
    4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋NL, 1),
    7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋NL, 1),
    18 := (⌊m'⌋N, 1),
    19 := (⌊H⌋N, 1),
    20 := (⌊m' * H⌋N, 1),
    1 := nlltape
      (formula-n Φ0 @ formula-n Φ1 @ formula-n Φ2 @ formula-n Φ3 @ formula-n Φ4 @
       formula-n Φ5 @ formula-n Φ6 @ formula-n Φ7 @ formula-n Φ8),
    91 := (⌊N⌋N, 1),
    92 := (⌊H⌋N, 1)]

lemma tm49 [transforms-intros]:
  assumes ttt = ttt47 + 24 + 3 * nlength N + 2 * nlength H
  shows transforms tm49 tps0 ttt tps49
  unfolding tm49-def
proof (tform)
  show 92 < length tps48
    using lentpsJ tps48-def k by (simp-all only: length-list-update)
  have tps48 ! 92 = tpsJ ! 92
    using tps48-def by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  then show tps48 ! 92 = (⌊0⌋N, 1)
    using tpsJ k canrepr-0 by simp
  show tps49 = tps48[92 := (⌊H⌋N, 1)]
    unfolding tps49-def tps48-def by (simp only: list-update-swap list-update-overwrite)
  show ttt = ttt47 + 14 + 3 * nlength N + (10 + 2 * nlength 0 + 2 * nlength H)
    using assms by simp
qed

definition tps50 ≡ tpsJ
  [11 := (⌊n⌋N, 1),
```

15 := ($\lfloor p\ n\rfloor_N$, *1*),
16 := ($\lfloor m\rfloor_N$, *1*),
17 := ($\lfloor T'\rfloor_N$, *1*),
4 := ($\lfloor$*map* ($\lambda t.$ *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* ($\lambda t.$ *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m'\rfloor_N$, *1*),
19 := ($\lfloor H\rfloor_N$, *1*),
20 := ($\lfloor m' * H\rfloor_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6 @ *formula-n* Φ_7 @ *formula-n* Φ_8),
91 := ($\lfloor N\rfloor_N$, *1*),
92 := ($\lfloor H\rfloor_N$, *1*),
93 := ($\lfloor Z\rfloor_N$, *1*)]

lemma *tm50* [*transforms-intros*]:
assumes *ttt* = *ttt47* + *34* + *3* ∗ *nlength N* + *2* ∗ *nlength H* + *2* ∗ *nlength Z*
shows *transforms tm50 tps0 ttt tps50*
unfolding *tm50-def*
proof (*tform*)
show *93* < *length tps49*
using *lentpsJ tps49-def k* **by** (*simp-all only*: *length-list-update*)
have *tps49* ! *93* = *tpsJ* ! *93*
using *tps49-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps49* ! *93* = ($\lfloor 0\rfloor_N$, *1*)
using *tpsJ k canrepr-0* **by** *simp*
show *tps50* = *tps49*[*93* := ($\lfloor Z\rfloor_N$, *1*)]
unfolding *tps50-def tps49-def* **by** (*simp only*: *list-update-swap list-update-overwrite*)
show *ttt* = *ttt47* + *24* + *3* ∗ *nlength N* + *2* ∗ *nlength H* + (*10* + *2* ∗ *nlength 0* + *2* ∗ *nlength Z*)
using *assms* **by** *simp*
qed

definition *tps51* ≡ *tpsJ*
[*11* := ($\lfloor n\rfloor_N$, *1*),
15 := ($\lfloor p\ n\rfloor_N$, *1*),
16 := ($\lfloor m\rfloor_N$, *1*),
17 := ($\lfloor T'\rfloor_N$, *1*),
4 := ($\lfloor$*map* ($\lambda t.$ *exc zs t* <#> *0*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* ($\lambda t.$ *exc zs t* <#> *1*) [*0*..<*Suc T'*]$\rfloor_{NL}$, *1*),
18 := ($\lfloor m'\rfloor_N$, *1*),
19 := ($\lfloor H\rfloor_N$, *1*),
20 := ($\lfloor m' * H\rfloor_N$, *1*),
1 := *nlltape*
(*formula-n* Φ_0 @ *formula-n* Φ_1 @ *formula-n* Φ_2 @ *formula-n* Φ_3 @ *formula-n* Φ_4 @
formula-n Φ_5 @ *formula-n* Φ_6 @ *formula-n* Φ_7 @ *formula-n* Φ_8),
91 := ($\lfloor N\rfloor_N$, *1*),
92 := ($\lfloor H\rfloor_N$, *1*),
93 := ($\lfloor Z\rfloor_N$, *1*),
94 := ($\lfloor T'\rfloor_N$, *1*)]

lemma *tm51* [*transforms-intros*]:
assumes *ttt* = *ttt47* + *48* + *3* ∗ *nlength N* + *2* ∗ *nlength H* + *2* ∗ *nlength Z* + *3* ∗ *nlength T'*
shows *transforms tm51 tps0 ttt tps51*
unfolding *tm51-def*
proof (*tform*)
show *17* < *length tps50 94* < *length tps50*
using *lentpsJ tps50-def k* **by** (*simp-all only*: *length-list-update*)
show *tps50* ! *17* = ($\lfloor T'\rfloor_N$, *1*)
using *tps50-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have *tps50* ! *94* = *tpsJ* ! *94*
using *tps50-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps50* ! *94* = ($\lfloor 0\rfloor_N$, *1*)
using *tpsJ k canrepr-0* **by** *simp*

```
  show tps51 = tps50[94 := (⌊T'⌋_N, 1)]
    unfolding tps51-def tps50-def by (simp only: list-update-swap list-update-overwrite)
  show ttt = ttt47 + 34 + 3 * nlength N + 2 * nlength H + 2 * nlength Z +
      (14 + 3 * (nlength T' + nlength 0))
    using assms by simp
qed

definition tps52 ≡ tpsJ
  [11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊m⌋_N, 1),
   17 := (⌊T'⌋_N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
   18 := (⌊m'⌋_N, 1),
   19 := (⌊H⌋_N, 1),
   20 := (⌊m' * H⌋_N, 1),
   1 := nlltape
      (formula-n Φ_0 @ formula-n Φ_1 @ formula-n Φ_2 @ formula-n Φ_3 @ formula-n Φ_4 @
       formula-n Φ_5 @ formula-n Φ_6 @ formula-n Φ_7 @ formula-n Φ_8),
   91 := (⌊N⌋_N, 1),
   92 := (⌊H⌋_N, 1),
   93 := (⌊Z⌋_N, 1),
   94 := (⌊T'⌋_N, 1),
   95 := (⌊formula-n ψ⌋_NLL, 1)]

lemma tm52 [transforms-intros]:
  assumes ttt = ttt47 + 58 + 3 * nlength N + 2 * nlength H + 2 * nlength Z +
    3 * nlength T' + 2 * nlllength (formula-n ψ)
  shows transforms tm52 tps0 ttt tps52
  unfolding tm52-def
proof (tform)
  show 95 < length tps51
    using lentpsJ tps51-def k by (simp-all only: length-list-update)
  have tps51 ! 95 = tpsJ ! 95
    using tps51-def by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  then have *: tps51 ! 95 = (⌊[]⌋, 1)
    using tpsJ k by simp
  then show tps51 ::: 95 = ⌊[]⌋
    by simp
  show clean-tape (tps51 ! 95)
    using * by simp
  show proper-symbols []
    by simp
  show proper-symbols (numlistlist (formula-n ψ))
    using proper-symbols-numlistlist by simp
  have tps52 = tps51[95 := (⌊formula-n ψ⌋_NLL, 1)]
    unfolding tps52-def tps51-def by (simp only: list-update-swap list-update-overwrite)
  then show tps52 = tps51[95 := (⌊numlistlist (formula-n ψ)⌋, 1)]
    using nllcontents-def by simp
  show ttt = ttt47 + 48 + 3 * nlength N + 2 * nlength H + 2 * nlength Z +
      3 * nlength T' + (8 + tps51 :#: 95 + 2 * length [] +
      Suc (2 * length (numlistlist (formula-n ψ))))
    using assms * nlllength-def by simp
qed

definition tps53 ≡ tpsJ
  [11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊m⌋_N, 1),
   17 := (⌊T'⌋_N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
```

```
    18 := (⌊m'⌋_N, 1),
    19 := (⌊H⌋_N, 1),
    20 := (⌊m' * H⌋_N, 1),
    1 := nlltape
      (formula-n Φ_0 @ formula-n Φ_1 @ formula-n Φ_2 @ formula-n Φ_3 @ formula-n Φ_4 @
       formula-n Φ_5 @ formula-n Φ_6 @ formula-n Φ_7 @ formula-n Φ_8),
    91 := (⌊N⌋_N, 1),
    92 := (⌊H⌋_N, 1),
    93 := (⌊Z⌋_N, 1),
    94 := (⌊T'⌋_N, 1),
    95 := (⌊formula-n ψ⌋_NLL, 1),
    96 := (⌊formula-n ψ'⌋_NLL, 1)]

lemma tm53 [transforms-intros]:
  assumes ttt = ttt47 + 68 + 3 * nlength N + 2 * nlength H + 2 * nlength Z + 3 * nlength T' +
    2 * nlllength (formula-n ψ) + 2 * length (numlistlist (formula-n ψ'))
  shows transforms tm53 tps0 ttt tps53
  unfolding tm53-def
proof (tform)
  show 96 < length tps52
    using lentpsJ tps52-def k by (simp-all only: length-list-update)
  have tps52 ! 96 = tpsJ ! 96
    using tps52-def by (simp only: length-list-update nth-list-update-eq nth-list-update-neq)
  then have *: tps52 ! 96 = (⌊[]⌋, 1)
    using tpsJ k by simp
  then show tps52 ::: 96 = ⌊[]⌋
    by simp
  show clean-tape (tps52 ! 96)
    using * by simp
  show proper-symbols []
    by simp
  show proper-symbols (numlistlist (formula-n ψ'))
    using proper-symbols-numlistlist by simp
  have tps53 = tps52[96 := (⌊formula-n ψ'⌋_NLL, 1)]
    unfolding tps53-def tps52-def by (simp only: list-update-swap list-update-overwrite)
  then show tps53 = tps52[96 := (⌊numlistlist (formula-n ψ')⌋, 1)]
    using nllcontents-def by simp
  show ttt = ttt47 + 58 + 3 * nlength N + 2 * nlength H + 2 * nlength Z + 3 * nlength T' +
      2 * nlllength (formula-n ψ) +
      (8 + tps52 :#: 96 + 2 * length [] + Suc (2 * length (numlistlist (formula-n ψ'))))
    using assms * nlllength-def by simp
qed

definition tps54 ≡ tpsJ
  [11 := (⌊n⌋_N, 1),
    15 := (⌊p n⌋_N, 1),
    16 := (⌊m⌋_N, 1),
    17 := (⌊T'⌋_N, 1),
    4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
    7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
    18 := (⌊m'⌋_N, 1),
    19 := (⌊H⌋_N, 1),
    20 := (⌊m' * H⌋_N, 1),
    1 := nlltape
      (formula-n Φ_0 @ formula-n Φ_1 @ formula-n Φ_2 @ formula-n Φ_3 @ formula-n Φ_4 @
       formula-n Φ_5 @ formula-n Φ_6 @ formula-n Φ_7 @ formula-n Φ_8),
    91 := (⌊N⌋_N, 1),
    92 := (⌊H⌋_N, 1),
    93 := (⌊Z⌋_N, 1),
    94 := (⌊T'⌋_N, 1),
    95 := (⌊formula-n ψ⌋_NLL, 1),
    96 := (⌊formula-n ψ'⌋_NLL, 1),
    97 := (⌊1⌋_N, 1)]
```

lemma *tm54* [*transforms-intros*]:
assumes *ttt* = *ttt47* + *80* + *3* ∗ *nlength N* + *2* ∗ *nlength H* + *2* ∗ *nlength Z* + *3* ∗ *nlength T′* +
2 ∗ *nlllength* (*formula-n ψ*) + *2* ∗ *nlllength* (*formula-n ψ′*)
shows *transforms tm54 tps0 ttt tps54*
unfolding *tm54-def*
proof (*tform*)
show *97* < *length tps53*
using *lentpsJ tps53-def k* **by** (*simp-all only*: *length-list-update*)
have *tps53* ! *97* = *tpsJ* ! *97*
using *tps53-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps53* ! *97* = (⌊*0*⌋$_N$, *1*)
using *tpsJ k canrepr-0* **by** *simp*
show *tps54* = *tps53*[*97* := (⌊*1*⌋$_N$, *1*)]
unfolding *tps54-def tps53-def* **by** (*simp only*: *list-update-swap list-update-overwrite*)
show *ttt* = *ttt47* + *68* + *3* ∗ *nlength N* + *2* ∗ *nlength H* + *2* ∗ *nlength Z* + *3* ∗ *nlength T′* +
2 ∗ *nlllength* (*formula-n ψ*) + *2* ∗ *length* (*numlistlist* (*formula-n ψ′*)) +
(*10* + *2* ∗ *nlength 0* + *2* ∗ *nlength 1*)
using *assms canrepr-1 nlllength-def* **by** *simp*
qed

definition *tps55* ≡ *tpsJ*
[*11* := (⌊*n*⌋$_N$, *1*),
15 := (⌊*p n*⌋$_N$, *1*),
16 := (⌊*m*⌋$_N$, *1*),
17 := (⌊*T′*⌋$_N$, *1*),
4 := (⌊*map* (λ*t*. *exc zs t* <#> *0*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
7 := (⌊*map* (λ*t*. *exc zs t* <#> *1*) [*0*..<*Suc T′*]⌋$_{NL}$, *1*),
18 := (⌊*m′*⌋$_N$, *1*),
19 := (⌊*H*⌋$_N$, *1*),
20 := (⌊*m′* ∗ *H*⌋$_N$, *1*),
1 := *nlltape* (*formula-n PHI*),
91 := (⌊*N*⌋$_N$, *1*),
92 := (⌊*H*⌋$_N$, *1*),
93 := (⌊*Z*⌋$_N$, *1*),
94 := (⌊*T′*⌋$_N$, *1*),
95 := (⌊*formula-n ψ*⌋$_{NLL}$, *1*),
96 := (⌊*formula-n ψ′*⌋$_{NLL}$, *1*),
97 := (⌊*1*⌋$_N$, *1*),
91 + *6* := (⌊*Suc T′*⌋$_N$, *1*),
91 + *3* := (⌊*0*⌋$_N$, *1*)]

definition *ttt55* ≡ *ttt47* + *80* + *3* ∗ *nlength N* + *2* ∗ *nlength H* + *2* ∗ *nlength Z* +
3 ∗ *nlength T′* + *2* ∗ *nlllength* (*formula-n ψ*) + *2* ∗ *nlllength* (*formula-n ψ′*) +
16114767 ∗ *2* ^ (*16* ∗ *Z*) ∗ *N* ^ *7*

lemma *ttt55*: *ttt55* ≤ *ttt47* + *2* ∗ *nlllength* (*formula-n ψ*) + *2* ∗ *nlllength* (*formula-n ψ′*) +
16114857 ∗ *2* ^ (*16* ∗ *Z*) ∗ *N* ^ *7*
proof −
have *nlength-le-ZN*: *y* ≤ *N* ⟹ *nlength y* ≤ *2* ^ (*16*∗*Z*)∗ *N* ^ *7* **for** *y*
proof −
assume *y* ≤ *N*
then have *nlength y* ≤ *N* ^ *7*
using *nlength-le-n linear-le-pow H-ge-3* **by** (*meson dual-order.trans zero-less-numeral*)
also have ... ≤ *2* ^ (*16*∗*Z*) ∗ *N* ^ *7*
by *simp*
finally show *?thesis* **.**
qed

have *3* ∗ *nlength N* ≤ *3* ∗ *2* ^ (*16*∗*Z*) ∗ *N* ^ *7*
using *nlength-le-ZN* **by** *simp*
moreover have *2* ∗ *nlength H* ≤ *2* ∗ *2* ^ (*16*∗*Z*) ∗ *N* ^ *7*
using *nlength-le-ZN*[*OF H-le-N*] **by** *simp*

moreover have $2 * nlength\ Z \leq 2 * 2 \,\hat{}\, (16{*}Z) * N \,\hat{}\, 7$
proof −
have $Z \leq N$
using *N-eq Z-def* **by** *simp*
then show *?thesis*
using *nlength-le-ZN* **by** *simp*
qed
moreover have $3 * nlength\ T' \leq 3 * 2 \,\hat{}\, (16{*}Z) * N \,\hat{}\, 7$
using *nlength-le-ZN*[*OF le-N*] **by** *simp*
moreover have $80 \leq 80 * 2 \,\hat{}\, (16{*}Z) * N \,\hat{}\, 7$
using *N-ge-1* **by** *simp*
ultimately have $ttt55 \leq ttt47 + 80 * 2 \,\hat{}\, (16{*}Z) * N \,\hat{}\, 7 + 3 * 2 \,\hat{}\, (16{*}Z) * N \,\hat{}\, 7 +$
$2 * 2 \,\hat{}\, (16{*}Z) * N \,\hat{}\, 7 + 2 * 2 \,\hat{}\, (16{*}Z) * N \,\hat{}\, 7 + 3 * 2 \,\hat{}\, (16{*}Z) * N \,\hat{}\, 7 +$
$2 * nlllength\ (formula\text{-}n\ \psi) + 2 * nlllength\ (formula\text{-}n\ \psi') +$
$16114767 * 2 \,\hat{}\, (16 * Z) * N \,\hat{}\, 7$
using *ttt55-def* **by** *linarith*
also have ... $= ttt47 + 2 * nlllength\ (formula\text{-}n\ \psi) + 2 * nlllength\ (formula\text{-}n\ \psi') +$
$16114857 * 2 \,\hat{}\, (16 * Z) * N \,\hat{}\, 7$
by *simp*
finally show *?thesis* **.**
qed

lemma *tm55* [*transforms-intros*]: *transforms tm55 tps0 ttt55 tps55*
unfolding *tm55-def*
proof (*tform*)
show $91 + 17 < length\ tps54$
using *lentpsJ tps54-def k* **by** (*simp-all only*: *length-list-update*)
let $?nss = formula\text{-}n\ \Phi_0 \ @\ formula\text{-}n\ \Phi_1 \ @\ formula\text{-}n\ \Phi_2 \ @\ formula\text{-}n\ \Phi_3 \ @\ formula\text{-}n\ \Phi_4 \ @$
$formula\text{-}n\ \Phi_5 \ @\ formula\text{-}n\ \Phi_6 \ @\ formula\text{-}n\ \Phi_7 \ @\ formula\text{-}n\ \Phi_8$
show $tps54\ !\ 1 = nlltape\ ?nss$
using *tps54-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show $tps54\ !\ 4 = (\lfloor map\ (\lambda t.\ exc\ zs\ t\ <\#>\ 0)\ [0..<Suc\ T']\rfloor_{NL},\ 1)$
using *tps54-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show $tps54\ !\ 7 = (\lfloor map\ (\lambda t.\ exc\ zs\ t\ <\#>\ 1)\ [0..<Suc\ T']\rfloor_{NL},\ 1)$
using *tps54-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show $tps54\ !\ 91 = (\lfloor N\rfloor_N,\ 1)$
using *tps54-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
have $tps54\ !\ 92 = (\lfloor H\rfloor_N,\ 1)$
using *tps54-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show $tps54\ !\ (91 + 1) = (\lfloor H\rfloor_N,\ 1)$
by *simp*
have $tps54\ !\ 93 = (\lfloor Z\rfloor_N,\ 1)$
using *tps54-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show $tps54\ !\ (91 + 2) = (\lfloor Z\rfloor_N,\ 1)$
by *simp*
have $tps54\ !\ 94 = (\lfloor T'\rfloor_N,\ 1)$
using *tps54-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show $tps54\ !\ (91 + 3) = (\lfloor T'\rfloor_N,\ 1)$
by *simp*
have $tps54\ !\ 95 = (\lfloor formula\text{-}n\ \psi\rfloor_{NLL},\ 1)$
using *tps54-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show $tps54\ !\ (91 + 4) = (\lfloor formula\text{-}n\ \psi\rfloor_{NLL},\ 1)$
by *simp*
have $tps54\ !\ 96 = (\lfloor formula\text{-}n\ \psi'\rfloor_{NLL},\ 1)$
using *tps54-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show $tps54\ !\ (91 + 5) = (\lfloor formula\text{-}n\ \psi'\rfloor_{NLL},\ 1)$
by *simp*
have $tps54\ !\ 97 = (\lfloor 1\rfloor_N,\ 1)$
using *tps54-def k lentpsJ* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show $tps54\ !\ (91 + 6) = (\lfloor 1\rfloor_N,\ 1)$
by *simp*
show $tps54\ !\ (91 + i) = (\lfloor[]\rfloor,\ 1)$ **if** $6 < i\ i < 17$ **for** i
proof −

have *tps54* ! (*91* + *i*) = *tpsJ* ! (*91* + *i*)
using *tps54-def that* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then show *tps54* ! (*91* + *i*) = ($\lfloor [] \rfloor$, *1*)
using *tpsJ k that* **by** *simp*
qed
have *tps55* = *tps54*
[*1* := *nlltape* (*formula-n PHI*),
91 + *6* := ($\lfloor Suc\ T' \rfloor_N$, *1*),
91 + *3* := ($\lfloor 0 \rfloor_N$, *1*)]
unfolding *tps55-def tps54-def* **by** (*simp only*: *list-update-swap list-update-overwrite*)
then show *tps55* = *tps54*
[*1* := *nlltape* (*?nss* @ *formula-n* (*PHI9*)),
91 + *6* := ($\lfloor Suc\ T' \rfloor_N$, *1*),
91 + *3* := ($\lfloor 0 \rfloor_N$, *1*)]
using *PHI-def formula-n-def* **by** *simp*
show *ttt55* = *ttt47* + *80* + *3* ∗ *nlength N* + *2* ∗ *nlength H* + *2* ∗ *nlength Z* +
3 ∗ *nlength* T' + *2* ∗ *nllength* (*formula-n* ψ) + *2* ∗ *nlllength* (*formula-n* ψ') +
16114767 ∗ *2* ^ (*16* ∗ *Z*) ∗ *N* ^ *7*
using *ttt55-def* **by** *simp*
qed

lemma *tps0-start-config*: (*0*, *tps0*) = *start-config 110 xs*
proof
show *fst* (*0*, *tps0*) = *fst* (*start-config 110 xs*)
using *start-config-def* **by** *simp*
let *?tps* = (λ*i*. *if i* = *0 then* ▷ *else if i* ≤ *length xs then xs* ! (*i* − *1*) *else* □, *0*) #
replicate (*110* − *1*) (λ*i*. *if i* = *0 then* ▷ *else* □, *0*)
have *tps0* = *?tps*
proof (*rule nth-equalityI*)
show *length tps0* = *length ?tps*
using *k* **by** *simp*
show *tps0* ! *j* = *?tps* ! *j* **if** *j* < *length tps0* **for** *j*
using *tps0 contents-def k that* **by** (*cases j* = *0*) *auto*
qed
then show *snd* (*0*, *tps0*) = *snd* (*start-config 110 xs*)
using *start-config-def* **by** *auto*
qed

lemma *tm55'*: *snd* (*execute tm55* (*start-config 110 xs*) *ttt55*) = *tps55*
using *tps0-start-config transforms-def transits-def tm55*
by (*smt* (*verit*, *best*) *execute-after-halting-ge prod.sel*(*1*) *prod.sel*(*2*))

definition *tpsK* ≡ *tpsJ*
[*91* := ($\lfloor N \rfloor_N$, *1*),
92 := ($\lfloor H \rfloor_N$, *1*),
93 := ($\lfloor Z \rfloor_N$, *1*),
94 := ($\lfloor T' \rfloor_N$, *1*),
95 := ($\lfloor formula\text{-}n\ \psi \rfloor_{NLL}$, *1*),
96 := ($\lfloor formula\text{-}n\ \psi' \rfloor_{NLL}$, *1*),
97 := ($\lfloor 1 \rfloor_N$, *1*),
91 + *6* := ($\lfloor Suc\ T' \rfloor_N$, *1*),
91 + *3* := ($\lfloor 0 \rfloor_N$, *1*)]

lemma *tpsK*: *97* < *j* ⟹ *j* < *110* ⟹ *tpsK* ! *j* = ($\lfloor [] \rfloor$, *1*)
using *tpsK-def tpsJ* **by** *simp*

lemma *lentpsK*: *length tpsK* = *110*
using *lentpsJ tpsK-def* **by** *simp*

lemma *tps55*: *tps55* = *tpsK*
[*11* := ($\lfloor n \rfloor_N$, *1*),
15 := ($\lfloor p\ n \rfloor_N$, *1*),
16 := ($\lfloor m \rfloor_N$, *1*),

```
  17 := (⌊T'⌋_N, 1),
  4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
  7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
  18 := (⌊m'⌋_N, 1),
  19 := (⌊H⌋_N, 1),
  20 := (⌊m' * H⌋_N, 1),
  1 := nlltape (formula-n PHI)]
 unfolding tps55-def tpsK-def by (simp only: list-update-swap)

definition tps56 ≡ tpsK
 [11 := (⌊n⌋_N, 1),
  15 := (⌊p n⌋_N, 1),
  16 := (⌊m⌋_N, 1),
  17 := (⌊T'⌋_N, 1),
  4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
  7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
  18 := (⌊m'⌋_N, 1),
  19 := (⌊H⌋_N, 1),
  20 := (⌊m' * H⌋_N, 1),
  1 := (⌊formula-n PHI⌋_NLL, 1)]

lemma tm56 [transforms-intros]:
 assumes ttt = ttt55 + tps55 :#: 1 + 2
 shows transforms tm56 tps0 ttt tps56
 unfolding tm56-def
proof (tform)
 show 1 < length tps55
  using lentpsJ tps55-def k by (simp-all only: length-list-update)
 have *: tps55 ! 1 = nlltape (formula-n PHI)
  using tps55 k lentpsK by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
 then show clean-tape (tps55 ! 1)
  using clean-tape-nllcontents by simp
 have tps55 ! 1 |#=| 1 = (⌊formula-n PHI⌋_NLL, 1)
  using * by simp
 then show tps56 = tps55[1 := tps55 ! 1 |#=| 1]
  unfolding tps56-def tps55 by (simp only: list-update-swap list-update-overwrite)
 show ttt = ttt55 + (tps55 :#: 1 + 2)
  using assms by simp
qed

definition tps57 ≡ tpsK
 [11 := (⌊n⌋_N, 1),
  15 := (⌊p n⌋_N, 1),
  16 := (⌊m⌋_N, 1),
  17 := (⌊T'⌋_N, 1),
  4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
  7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
  18 := (⌊m'⌋_N, 1),
  19 := (⌊H⌋_N, 1),
  20 := (⌊m' * H⌋_N, 1),
  1 := nlltape (formula-n PHI),
  109 := nlltape (formula-n PHI)]

lemma tm57 [transforms-intros]:
 assumes ttt = ttt55 + tps55 :#: 1 + 2 + Suc (nlllength (formula-n PHI))
 shows transforms tm57 tps0 ttt tps57
 unfolding tm57-def
proof (tform)
 show 1 < length tps56 109 < length tps56
  using lentpsK tps56-def k by (simp-all only: length-list-update)
 have *: tps56 ! 1 = (⌊formula-n PHI⌋_NLL, 1)
  using tps56-def k lentpsK by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
 let ?n = nlllength (formula-n PHI)
```

show *rneigh* (*tps56* ! *1*) {*0*} *?n*
proof (*rule rneighI*)
show (*tps56* ::: *1*) (*tps56* :#: *1* + *nlllength* (*formula-n PHI*)) $\in$ {*0*}
using *proper-symbols-numlistlist nllcontents-def* $*$ *contents-outofbounds nlllength-def*
by *simp*
have $\bigwedge n'.\ n' <$ *?n* $\Longrightarrow$ (*tps56* ::: *1*) (*1* + n') > *0*
using *proper-symbols-numlistlist nllcontents-def* $*$ *contents-inbounds nlllength-def*
by *fastforce*
then show $\bigwedge n'.\ n' <$ *?n* $\Longrightarrow$ (*tps56* ::: *1*) (*tps56* :#: *1* + n') $\notin$ {*0*}
using $*$ **by** *simp*
qed
have *tps56* ! *109* = *tpsK* ! *109*
using *tps56-def* **by** (*simp only*: *length-list-update nth-list-update-eq nth-list-update-neq*)
then have $**$: *tps56* ! *109* = ($\lfloor$[]$\rfloor$, *1*)
using *tpsK k* **by** *simp*
have *implant* ($\lfloor$*formula-n PHI*$\rfloor_{NLL}$, *1*) ($\lfloor$[]$\rfloor$, *1*) *?n* =
($\lfloor$[] @
take (*nlllength* (*formula-n PHI*))
(*drop* (*1* − *1*) (*numlistlist* (*formula-n PHI*)))$\rfloor$,
Suc (*length* []) + *nlllength* (*formula-n PHI*))
using *implant-contents*[*of 1 ?n numlistlist* (*formula-n PHI*) []] *nlllength-def nllcontents-def*
by *simp*
then have *implant* ($\lfloor$*formula-n PHI*$\rfloor_{NLL}$, *1*) ($\lfloor$[]$\rfloor$, *1*) *?n* =
($\lfloor$*take ?n* (*numlistlist* (*formula-n PHI*))$\rfloor$, *Suc ?n*)
by *simp*
also have ... = ($\lfloor$*numlistlist* (*formula-n PHI*)$\rfloor$, *Suc ?n*)
using *nlllength-def* **by** *simp*
also have ... = ($\lfloor$*formula-n PHI*$\rfloor_{NLL}$, *Suc ?n*)
using *nllcontents-def* **by** *simp*
finally have *implant* ($\lfloor$*formula-n PHI*$\rfloor_{NLL}$, *1*) ($\lfloor$[]$\rfloor$, *1*) *?n* = ($\lfloor$*formula-n PHI*$\rfloor_{NLL}$, *Suc ?n*) **.**
then have *implant* (*tps56* ! *1*) (*tps56* ! *109*) *?n* = ($\lfloor$*formula-n PHI*$\rfloor_{NLL}$, *Suc ?n*)
using $*$ $**$ **by** *simp*
then have *implant* (*tps56* ! *1*) (*tps56* ! *109*) *?n* = *nlltape* (*formula-n PHI*)
by *simp*
moreover have *tps56* ! *1* |+| *nlllength* (*formula-n PHI*) = *nlltape* (*formula-n PHI*)
using $*$ **by** *simp*
moreover have *tps57* = *tps56*
[*1* := *nlltape* (*formula-n PHI*),
109 := *nlltape* (*formula-n PHI*)]
unfolding *tps57-def tps56-def* **by** (*simp only*: *list-update-swap*[*of 1*] *list-update-overwrite*)
ultimately show *tps57* = *tps56*
[*1* := *tps56* ! *1* |+| *nlllength* (*formula-n PHI*),
109 := *implant* (*tps56* ! *1*) (*tps56* ! *109*) (*nlllength* (*formula-n PHI*))]
by *simp*
show *ttt* = *ttt55* + *tps55* :#: *1* + *2* + *Suc* (*nlllength* (*formula-n PHI*))
using *assms* **by** *simp*
qed

definition *tps58* $\equiv$ *tpsK*
[*11* := ($\lfloor n\rfloor_N$, *1*),
15 := ($\lfloor p\ n\rfloor_N$, *1*),
16 := ($\lfloor m\rfloor_N$, *1*),
17 := ($\lfloor T'\rfloor_N$, *1*),
4 := ($\lfloor$*map* ($\lambda t.$ *exc zs t* <#> *0*) [*0*..<*Suc* T']$\rfloor_{NL}$, *1*),
7 := ($\lfloor$*map* ($\lambda t.$ *exc zs t* <#> *1*) [*0*..<*Suc* T']$\rfloor_{NL}$, *1*),
18 := ($\lfloor m'\rfloor_N$, *1*),
19 := ($\lfloor H\rfloor_N$, *1*),
20 := ($\lfloor m' * H\rfloor_N$, *1*),
1 := ($\lfloor$[]$\rfloor$, *1*),
109 := *nlltape* (*formula-n PHI*)]

lemma *tm58* [*transforms-intros*]:
assumes *ttt* = *ttt55* + *9* + *tps55* :#: *1* + *3* $*$ *nlllength* (*formula-n PHI*) + *tps57* :#: *1*

```
  shows transforms tm58 tps0 ttt tps58
  unfolding tm58-def
proof (tform)
  show 1 < length tps57
    using lentpsK tps57-def k by (simp-all only: length-list-update)
  let ?zs = numlistlist (formula-n PHI)
  show proper-symbols ?zs
    using proper-symbols-numlistlist by simp
  have tps57 ! 1 = nlltape (formula-n PHI)
    using tps57-def k lentpsK by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
  then have tps57 ! 1 = (⌊numlistlist (formula-n PHI)⌋, Suc (nlllength (formula-n PHI)))
    using nlllength-def nllcontents-def by auto
  then show tps57 ::: 1 = ⌊numlistlist (formula-n PHI)⌋
    by simp
  show tps58 = tps57[1 := (⌊[]⌋, 1)]
    unfolding tps58-def tps57-def by (simp only: list-update-swap list-update-overwrite)
  show ttt = ttt55 + tps55 :#: 1 + 2 + Suc (nlllength (formula-n PHI)) +
      (tps57 :#: 1 + 2 ∗ length (numlistlist (formula-n PHI)) + 6)
    using assms nlllength-def by simp
qed

definition tps59 ≡ tpsK
  [11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊m⌋_N, 1),
   17 := (⌊T'⌋_N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
   18 := (⌊m'⌋_N, 1),
   19 := (⌊H⌋_N, 1),
   20 := (⌊m' ∗ H⌋_N, 1),
   1 := (⌊[]⌋, 1),
   109 := (⌊formula-n PHI⌋_NLL, 1)]

lemma tm59 [transforms-intros]:
  assumes ttt = ttt55 + 11 + tps55 :#: 1 + 3 ∗ nlllength (formula-n PHI) + tps57 :#: 1 + tps58 :#: 109
  shows transforms tm59 tps0 ttt tps59
  unfolding tm59-def
proof (tform)
  show 109 < length tps58
    using lentpsK tps58-def k by (simp-all only: length-list-update)
  have ∗: tps58 ! 109 = nlltape (formula-n PHI)
    using tps58-def k lentpsK by (simp only: length-list-update nth-list-update-neq nth-list-update-eq)
  then show clean-tape (tps58 ! 109)
    by (simp add: clean-tape-nllcontents)
  have tps58 ! 109 |#=| 1 = (⌊formula-n PHI⌋_NLL, 1)
    using ∗ by simp
  then show tps59 = tps58[109 := tps58 ! 109 |#=| 1]
    unfolding tps59-def tps58-def by (simp only: list-update-swap list-update-overwrite)
  show ttt = ttt55 + 9 + tps55 :#: 1 + 3 ∗ nlllength (formula-n PHI) + tps57 :#: 1 +
      (tps58 :#: 109 + 2)
    using assms by simp
qed

definition tps60 ≡ tpsK
  [11 := (⌊n⌋_N, 1),
   15 := (⌊p n⌋_N, 1),
   16 := (⌊m⌋_N, 1),
   17 := (⌊T'⌋_N, 1),
   4 := (⌊map (λt. exc zs t <#> 0) [0..<Suc T']⌋_NL, 1),
   7 := (⌊map (λt. exc zs t <#> 1) [0..<Suc T']⌋_NL, 1),
   18 := (⌊m'⌋_N, 1),
   19 := (⌊H⌋_N, 1),
```

20 := (⌊*m*′ ∗ *H*⌋$_N$, *1*),
1 := (⌊[]⌋, *1*),
109 := (⌊*formula-n PHI*⌋$_{NLL}$, *1*),
109 := (⌊*numlistlist* (*formula-n PHI*)⌋,
Suc (*length* (*numlistlist* (*formula-n PHI*)))),
1 := (⌊*binencode* (*numlistlist* (*formula-n PHI*))⌋,
Suc (*2* ∗ *length* (*numlistlist* (*formula-n PHI*))))]

lemma *tm60*:
assumes *ttt* = *ttt55* + *12* + *tps55* :#: *1* + *12* ∗ *nlllength* (*formula-n PHI*) + *tps57* :#: *1* + *tps58* :#: *109*
shows *transforms tm60 tps0 ttt tps60*
unfolding *tm60-def*
proof (*tform*)
show *109* < *length tps59 1* < *length tps59*
using *lentpsK tps59-def k* **by** (*simp-all only*: *length-list-update*)
let *?zs* = *numlistlist* (*formula-n PHI*)
show *binencodable ?zs*
using *proper-symbols-numlistlist symbols-lt-numlistlist* **by** *fastforce*
show *tps59* ! *109* = (⌊*numlistlist* (*formula-n PHI*)⌋, *1*)
using *tps59-def k lentpsK nllcontents-def*
by (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show *tps59* ! *1* = (⌊[]⌋, *1*)
using *tps59-def k lentpsK* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
show *tps60* ≡ *tps59*
[*109* := (⌊*numlistlist* (*formula-n PHI*)⌋,
Suc (*length* (*numlistlist* (*formula-n PHI*)))),
1 := (⌊*binencode* (*numlistlist* (*formula-n PHI*))⌋,
Suc (*2* ∗ *length* (*numlistlist* (*formula-n PHI*))))]
unfolding *tps60-def tps59-def* **by** (*simp only*: *list-update-swap list-update-overwrite*)
show *ttt* = *ttt55* + *11* + *tps55* :#: *1* + *3* ∗ *nlllength* (*formula-n PHI*) + *tps57* :#: *1* +
tps58 :#: *109* + (*9* ∗ *length* (*numlistlist* (*formula-n PHI*)) + *1*)
using *assms nlllength-def* **by** *simp*
qed

definition *ttt60* ≡ *16* ∗ *ttt55*

lemma *tm60*′: *transforms tm60 tps0 ttt60 tps60*
proof −
have *tps55-head-1*: *tps55* :#: *1* ≤ *ttt55*
proof −
have ∗: (*1*::*nat*) < *110*
using *k* **by** *simp*
show *?thesis*
using *head-pos-le-time*[*OF tm55-tm* ∗, *of xs ttt55*] *tm55*′ *k* **by** *simp*
qed

let *?ttt* = *ttt55* + *12* + *tps55* :#: *1* + *12* ∗ *nlllength* (*formula-n PHI*) + *tps57* :#: *1* + *tps58* :#: *109*
have *55*: *tps55* :#: *1* = *Suc* (*nlllength* (*formula-n PHI*))
proof −
have *tps55* ! *1* = *nlltape* (*formula-n PHI*)
using *tps55 k lentpsK* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show *?thesis*
by *simp*
qed
moreover have *tps57* :#: *1* = *Suc* (*nlllength* (*formula-n PHI*))
proof −
have *tps57* ! *1* = *nlltape* (*formula-n PHI*)
using *tps57-def k lentpsK* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show *?thesis*
by *simp*
qed
moreover have *tps58* :#: *109* = *Suc* (*nlllength* (*formula-n PHI*))
proof −

have *tps58* ! *109* = *nlltape* (*formula-n PHI*)
using *tps58-def k lentpsK* **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
then show *?thesis*
by *simp*
qed
ultimately have *?ttt* = *ttt55* + *12* + *3* ∗ (*Suc* (*nlllength* (*formula-n PHI*))) + *12* ∗ *nlllength* (*formula-n PHI*)
by *simp*
also have ... = *ttt55* + *15* + *15* ∗ (*nlllength* (*formula-n PHI*))
by *simp*
also have ... = *ttt55* + *15* ∗ (*Suc* (*nlllength* (*formula-n PHI*)))
by *simp*
also have ... ≤ *ttt55* + *15* ∗ *ttt55*
using *tps55-head-1 55* **by** *simp*
also have ... = *16* ∗ *ttt55*
by *simp*
finally have *?ttt* ≤ *16* ∗ *ttt55* **.**
then show *?thesis*
using *tm60 transforms-monotone ttt60-def* **by** *simp*
qed

lemma *tm60-start-config*: *transforms tm60* (*snd* (*start-config 110* (*string-to-symbols x*))) *ttt60 tps60*
using *tm60′ tps0-start-config* **by** (*metis prod.sel*(*2*))

end

end

The time bound *ttt60* formally depends on the string x. But we need a bound depending only on the length.

context *reduction-sat*
begin

definition *T60* :: *nat* ⇒ *nat* **where**
T60 nn ≡ *reduction-sat-x.ttt60 M G p* (*replicate nn True*)

lemma *T60*:
fixes *x* :: *string*
shows *T60* (*length x*) = *reduction-sat-x.ttt60 M G p x*
proof −
interpret *x*: *reduction-sat-x L M G T p x*
by (*simp add*: *reduction-sat-axioms reduction-sat-x.intro*)

define *tpsx* :: *tape list* **where** *tpsx* = *snd* (*start-config 110* (*x.xs*))
have *x1*: *110* = *length tpsx*
using *start-config-def tpsx-def* **by** *auto*
have *x2*: *tpsx* ! *0* = (⌊*x.xs*⌋, *0*)
using *start-config-def tpsx-def* **by** *auto*
have *x3*: ⋀*i*. *0* < *i* ⟹ *i* < *110* ⟹ *tpsx* ! *i* = (⌊[]⌋, *0*)
using *start-config-def tpsx-def* **by** *auto*

let *?y* = *replicate* (*length x*) *True*
interpret *y*: *reduction-sat-x L M G T p ?y*
by (*simp add*: *reduction-sat-axioms reduction-sat-x.intro*)
define *tpsy* :: *tape list* **where** *tpsy* = *snd* (*start-config 110* (*y.xs*))
have *y1*: *110* = *length tpsy*
using *start-config-def tpsy-def* **by** *auto*
have *y2*: *tpsy* ! *0* = (⌊*y.xs*⌋, *0*)
using *start-config-def tpsy-def* **by** *auto*
have *y3*: ⋀*i*. *0* < *i* ⟹ *i* < *110* ⟹ *tpsy* ! *i* = (⌊[]⌋, *0*)
using *start-config-def tpsy-def* **by** *auto*

have *m*: *x.m* = *y.m*

using *x.m-def y.m-def* **by** *simp*
have T': $x.T' = y.T'$
using $x.T'$*-def* $y.T'$*-def m* **by** *simp*
have m': $x.m' = y.m'$
using $x.m'$*-def* $y.m'$*-def* T' **by** *simp*
have N: $x.N = y.N$
using *x.N-eq y.N-eq* T' **by** *simp*

have *x.ttt31* = *y.ttt31*
using *x.ttt31-def*[*OF x1 x2 x3*] *y.ttt31-def*[*OF y1 y2 y3*] T' *m* m' **by** *simp*
then have *x.ttt47* = *y.ttt47*
using *x.ttt47-def*[*OF x1 x2 x3*] *y.ttt47-def*[*OF y1 y2 y3*] T' *m* m' **by** *simp*
then have *x.ttt55* = *y.ttt55*
using *x.ttt55-def*[*OF x1 x2 x3*] *y.ttt55-def*[*OF y1 y2 y3*] T' *m* m' *N* **by** *simp*
then have *x.ttt60* = *y.ttt60*
using *x.ttt60-def*[*OF x1 x2 x3*] *y.ttt60-def*[*OF y1 y2 y3*] **by** *simp*
then show *T60* (*length x*) = *reduction-sat-x.ttt60 M G p x*
unfolding *T60-def* **by** *simp*
qed

lemma *poly-T60*: *big-oh-poly T60*
proof −
define *fN* :: *nat* ⇒ *nat* **where**
fN = (λ*nn. H* ∗ (*2* ∗ *nn* + *2* ∗ *p nn* + *3* + *TT* (*2* ∗ *nn* + *2* ∗ *p nn* + *2*)))
define *f* **where**
f = (λ*nn. 16* ∗ ((*32* ∗ *d-polynomial p* + *364343*) ∗ *H* ^ *6* ∗ *fN nn* ^ *4* +
2 ∗ *nlllength* (*formula-n* ψ) + *2* ∗ *nlllength* (*formula-n* ψ′) + *16114857* ∗ *2* ^ (*16* ∗ *Z*) ∗ *fN nn* ^ *7*))
have *T60-upper*: *T60 nn* ≤ *f nn* **for** *nn*
proof −
define *y* **where** *y* = *replicate nn True*
then have *leny*: *length y* = *nn*
by *simp*
interpret *y*: *reduction-sat-x - - - - - y*
by (*simp add*: *reduction-sat-axioms reduction-sat-x.intro*)
define *tps0* :: *tape list* **where** *tps0* = *snd* (*start-config 110 y.xs*)
have *1*: *110* = *length tps0*
using *start-config-def tps0-def* **by** *auto*
have *2*: *tps0* ! *0* = (⌊*y.xs*⌋, *0*)
using *start-config-def tps0-def* **by** *auto*
have *3*: ⋀*i. 0* < *i* ⟹ *i* < *110* ⟹ *tps0* ! *i* = (⌊[]⌋, *0*)
using *start-config-def tps0-def* **by** *auto*

have *T60 nn* = *y.ttt60*
by (*simp add*: *y-def T60-def*)
also have ... ≤ *16* ∗ *y.ttt55*
using *y.ttt60-def*[*OF 1 2 3*] **by** *simp*
also have ... ≤ *16* ∗ (*y.ttt47* + *2* ∗ *nlllength* (*formula-n* ψ) + *2* ∗ *nlllength* (*formula-n* ψ′) + *16114857* ∗ *2* ^ (*16* ∗ *Z*) ∗ *y.N* ^ *7*)
using *y.ttt55*[*OF 1 2 3*] **by** *simp*
also have ... ≤ *16* ∗ ((*32* ∗ *d-polynomial p* + *364343*) ∗ *H* ^ *6* ∗ *y.N* ^ *4* +
2 ∗ *nlllength* (*formula-n* ψ) + *2* ∗ *nlllength* (*formula-n* ψ′) + *16114857* ∗ *2* ^ (*16* ∗ *Z*) ∗ *y.N* ^ *7*)
using *y.ttt47*[*OF 1 2 3*] **by** *simp*
also have ... = *16* ∗ ((*32* ∗ *d-polynomial p* + *364343*) ∗ *H* ^ *6* ∗ *fN nn* ^ *4* +
2 ∗ *nlllength* (*formula-n* ψ) + *2* ∗ *nlllength* (*formula-n* ψ′) + *16114857* ∗ *2* ^ (*16* ∗ *Z*) ∗ *fN nn* ^ *7*)
proof −
have *y.N* = *fN nn*
using *y.N-eq* $y.T'$*-def y.m-def y-def fN-def* **by** *simp*
then show *?thesis*
by *simp*
qed
finally have *T60 nn* ≤ *16* ∗ ((*32* ∗ *d-polynomial p* + *364343*) ∗ *H* ^ *6* ∗ *fN nn* ^ *4* +
2 ∗ *nlllength* (*formula-n* ψ) + *2* ∗ *nlllength* (*formula-n* ψ′) + *16114857* ∗ *2* ^ (*16* ∗ *Z*) ∗ *fN nn* ^ *7*) **.**
then show *?thesis*

using *f-def* **by** *simp*
qed

have ∗: *big-oh-poly fN*
proof −
have *5*: *big-oh-poly p*
using *big-oh-poly-polynomial*[*OF p*] **by** *simp*
have *6*: *big-oh-poly TT*
using *T big-oh-poly-le TT-le* **by** *simp*
have *big-oh-poly* (*λnn. 2 ∗ p nn + 2*)
using *5 big-oh-poly-sum big-oh-poly-prod big-oh-poly-const*
by *presburger*
moreover have *big-oh-poly* (*λnn. 2 ∗ nn*)
using *big-oh-poly-prod big-oh-poly-const big-oh-poly-id* **by** *simp*
ultimately have *big-oh-poly* (*λnn. 2 ∗ nn + 2 ∗ p nn + 2*)
using *big-oh-poly-sum* **by** *fastforce*
then have *big-oh-poly* (*TT* ∘ (*λnn. 2 ∗ nn + 2 ∗ p nn + 2*))
using *big-oh-poly-composition*[*OF - 6*] **by** *simp*
moreover have *TT* ∘ (*λnn. 2 ∗ nn + 2 ∗ p nn + 2*) = (*λnn. TT* (*2 ∗ nn + 2 ∗ p nn + 2*))
by *auto*
ultimately have *big-oh-poly* (*λnn. TT* (*2 ∗ nn + 2 ∗ p nn + 2*))
by *simp*
moreover have *big-oh-poly* (*λnn. 2 ∗ nn + 2 ∗ p nn + 3*)
using *5 big-oh-poly-prod big-oh-poly-const big-oh-poly-sum big-oh-poly-id* **by** *simp*
ultimately have *big-oh-poly* (*λnn. 2 ∗ nn + 2 ∗ p nn + 3 + TT* (*2 ∗ nn + 2 ∗ p nn + 2*))
using *big-oh-poly-sum* **by** *simp*
then have *big-oh-poly* (*λnn. H ∗* (*2 ∗ nn + 2 ∗ p nn + 3 + TT* (*2 ∗ nn + 2 ∗ p nn + 2*)))
using *big-oh-poly-prod big-oh-poly-const* **by** *simp*
then show *?thesis*
using *fN-def* **by** *simp*
qed
then have *big-oh-poly* (*λn. fN n* ^ *4*)
using *big-oh-poly-pow* **by** *simp*
moreover have *big-oh-poly* (*λn.* (*32 ∗ d-polynomial p + 364343*) *∗ H* ^ *6*)
using *big-oh-poly-prod big-oh-poly-const big-oh-poly-sum* **by** *simp*
ultimately have *big-oh-poly* (*λn.* (*32 ∗ d-polynomial p + 364343*) *∗ H* ^ *6 ∗ fN n* ^ *4*)
using *big-oh-poly-prod* **by** *simp*
moreover have *big-oh-poly* (*λn. 2 ∗ nlllength* (*formula-n ψ*) + *2 ∗ nlllength* (*formula-n ψ′*) + *16114857 ∗ 2* ^ (*16 ∗ Z*) *∗ fN n* ^ *7*)
using *big-oh-poly-pow ∗ big-oh-poly-sum big-oh-poly-prod big-oh-poly-const*
by *simp*
ultimately have *big-oh-poly* (*λn.* ((*32 ∗ d-polynomial p + 364343*) *∗ H* ^ *6 ∗ fN n* ^ *4*) +
(*2 ∗ nlllength* (*formula-n ψ*) + *2 ∗ nlllength* (*formula-n ψ′*) + *16114857 ∗ 2* ^ (*16 ∗ Z*) *∗ fN n* ^ *7*))
using *big-oh-poly-sum* **by** *simp*
moreover have (*λn.* ((*32 ∗ d-polynomial p + 364343*) *∗ H* ^ *6 ∗ fN n* ^ *4*) +
(*2 ∗ nlllength* (*formula-n ψ*) + *2 ∗ nlllength* (*formula-n ψ′*) + *16114857 ∗ 2* ^ (*16 ∗ Z*) *∗ fN n* ^ *7*)) =
(*λn.* (*32 ∗ d-polynomial p + 364343*) *∗ H* ^ *6 ∗ fN n* ^ *4* +
2 ∗ nlllength (*formula-n ψ*) + *2 ∗ nlllength* (*formula-n ψ′*) + *16114857 ∗ 2* ^ (*16 ∗ Z*) *∗ fN n* ^ *7*)
by *auto*
ultimately have *big-oh-poly* (*λn.* (*32 ∗ d-polynomial p + 364343*) *∗ H* ^ *6 ∗ fN n* ^ *4* +
2 ∗ nlllength (*formula-n ψ*) + *2 ∗ nlllength* (*formula-n ψ′*) + *16114857 ∗ 2* ^ (*16 ∗ Z*) *∗ fN n* ^ *7*)
by *simp*
then have *big-oh-poly f*
using *f-def big-oh-poly-prod big-oh-poly-const* **by** *blast*
then show *big-oh-poly T60*
using *T60-upper big-oh-poly-le* **by** *simp*
qed

This is the function, in terms of bit strings, that maps x to Φ.

definition *freduce* :: *string* ⇒ *string* (‹f_{reduce}›) **where**
f_{reduce} *x* ≡ *formula-to-string* (*reduction-sat-x.PHI M G p x*)

The function f_{reduce} many-one reduces L to `SAT`.

lemma *x-in-L*: $x \in L \longleftrightarrow f_{reduce}\ x \in SAT$
proof
  interpret *x*: *reduction-sat-x*
    by (*simp add*: *reduction-sat-axioms reduction-sat-x.intro*)
  show $x \in L \Longrightarrow f_{reduce}\ x \in SAT$
    using *freduce-def SAT-def x.L-iff-satisfiable* **by** *auto*
  show $f_{reduce}\ x \in SAT \Longrightarrow x \in L$
  proof −
    assume $f_{reduce}\ x \in SAT$
    then obtain *phi* **where**
      phi: *satisfiable phi* $f_{reduce}\ x$ = *formula-to-string phi*
      using *SAT-def freduce-def* **by** *auto*

    have *formula-to-string* (*reduction-sat-x.PHI M G p x*) = *formula-to-string phi*
      using *phi*(*2*) *freduce-def* **by** *simp*
    then have *reduction-sat-x.PHI M G p x* = *phi*
      using *formula-to-string-inj* **by** *simp*
    then have *satisfiable* (*reduction-sat-x.PHI M G p x*)
      using *phi*(*1*) **by** *simp*
    then show $x \in L$
      using *x.L-iff-satisfiable* **by** *simp*
  qed
qed

The Turing machine *tm60* computes f_{reduce} with time bound $T60$.

lemma *computes-in-time-tm60*: *computes-in-time 110 tm60* f_{reduce} *T60*
proof
  fix *x* :: *string*

  interpret *x*: *reduction-sat-x - - - - - x*
    by (*simp add*: *reduction-sat-axioms reduction-sat-x.intro*)

  have *binencodable* (*numlistlist* (*formula-n x.PHI*))
    by (*metis One-nat-def Suc-1 Suc-leI le-refl proper-symbols-numlistlist symbols-lt-numlistlist*)
  then have ∗: *bit-symbols* (*binencode* (*numlistlist* (*formula-n x.PHI*)))
    using *bit-symbols-binencode* **by** *simp*

  define *tps0* :: *tape list* **where** *tps0* = *snd* (*start-config 110 x.xs*)
  have *1*: *110* = *length tps0*
    using *start-config-def tps0-def* **by** *auto*
  have *2*: *tps0* ! *0* = (⌊*x.xs*⌋, *0*)
    using *start-config-def tps0-def* **by** *auto*
  have *3*: $\bigwedge i.\ 0 < i \Longrightarrow i < 110 \Longrightarrow$ *tps0* ! *i* = (⌊[]⌋, *0*)
    using *start-config-def tps0-def* **by** *auto*
  let *?tps* = *x.tps60 tps0*
  have *length ?tps* = *110*
    using *x.tps60-def*[*OF 1 2 3*] *x.lentpsK*[*OF 1 2 3*] **by** (*simp-all only*: *length-list-update*)
  then have *?tps* ! *1* = (⌊*binencode* (*numlistlist* (*formula-n* (*x.PHI*)))⌋,
      Suc (*2* ∗ *length* (*numlistlist* (*formula-n x.PHI*))))
    using *x.tps60-def*[*OF 1 2 3*] **by** (*simp only*: *length-list-update nth-list-update-neq nth-list-update-eq*)
  then have *?tps* ::: *1* = ⌊*binencode* (*numlistlist* (*formula-n x.PHI*))⌋
    by *simp*
  also have ... = *string-to-contents* (*symbols-to-string* (*binencode* (*numlistlist* (*formula-n x.PHI*))))
    using *bit-symbols-to-contents*[*OF* ∗] **by** *simp*
  also have ... = *string-to-contents* ($f_{reduce}\ x$)
    using *freduce-def* **by** *auto*
  finally have ∗∗: *?tps* ::: *1* = *string-to-contents* ($f_{reduce}\ x$) **.**

  have *transforms tm60 tps0 x.ttt60 ?tps*
    using *tps0-def x.tm60-start-config*[*OF 1 2 3*] **by** *simp*
  then have *transforms tm60* (*snd* (*start-config 110 x.xs*)) (*T60* (*length x*)) *?tps*
    using *T60 tps0-def* **by** *simp*

then show $\exists$ *tps*.
tps ::: *1* = *string-to-contents* (f_{reduce} *x*) $\wedge$
transforms tm60 (*snd* (*start-config 110* (*string-to-symbols x*))) (*T60* (*length x*)) *tps*
using ∗∗ **by** *auto*
qed

Since $T60$ is bounded by a polynomial, the previous three lemmas imply that L is polynomial-time many-one reducible to `SAT`.

lemma *L-reducible-SAT*: $L \leq_p$ *SAT*
using *reducible-def tm60-tm poly-T60 computes-in-time-tm60 x-in-L* **by** *fastforce*

end

In the locale *reduction-sat* the language L was chosen arbitrarily with properties that we have proven $\mathcal{NP}$ languages have. So we can now show that `SAT` is $\mathcal{NP}$-hard.

theorem *NP-hard-SAT*:
assumes $L \in \mathcal{NP}$
shows $L \leq_p$ *SAT*
proof −
obtain *M G T p* **where**
T: *big-oh-poly T* **and**
p: *polynomial p* **and**
tm-M: *turing-machine 2 G M* **and**
oblivious-M: *oblivious M* **and**
T-halt: $\bigwedge y$. *bit-symbols y* $\Longrightarrow$ *fst* (*execute M* (*start-config 2 y*) (*T* (*length y*))) = *length M* **and**
cert: $\bigwedge x$. $x \in L \longleftrightarrow$ ($\exists$ *u*. *length u* = *p* (*length x*) $\wedge$ *execute M* (*start-config 2* $\langle x; u\rangle$) (*T* (*length* $\langle x; u\rangle$))) <.>
1 = **1**)
using *NP-imp-oblivious-2tape*[*OF assms*] **by** *metis*

interpret *red*: *reduction-sat L M G T p*
using *T p tm-M oblivious-M T-halt cert reduction-sat.intro* **by** *simp*

show *?thesis*
using *red.L-reducible-SAT* **by** *simp*
qed

8.4 SAT is $\mathcal{NP}$-complete

The time has come to reap the fruits of our labor and show that `SAT` is $\mathcal{NP}$-complete.

theorem *NP-complete-SAT*: *NP-complete SAT*
using *NP-hard-SAT SAT-in-NP NP-complete-def* **by** *simp*

end

Bibliography

[1] Scott Aaronson. Complexity zoo. https://complexityzoo.net/Complexity_Zoo.

[2] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2006.

[3] Robert S. Boyer, J Strother Moore, and Matt Kaufmann. ACL2. https://www.cs.utexas.edu/users/moore/acl2/.

[4] Stephen A. Cook. The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing*, 1971.

[5] Christian Dalvit and René Thiemann. A verified translation of multitape turing machines into singletape turing machines. *Archive of Formal Proofs*, November 2022. https://isa-afp.org/entries/Multitape_To_Singletape_TM.html, Formal proof development.

[6] Yannick Forster and Gert Smolka. Weak call-by-value lambda calculus as a model of computation in Coq. In *ITP*, 2017.

[7] Lennard Gäher and Fabian Kunze. Mechanising complexity theory: The Cook-Levin theorem in Coq. In *ITP*, 2021.

[8] Ruben Gamboa and John R. Cowles. A mechanical proof of the Cook-Levin theorem. In *TPHOLs*, 2004.

[9] Juris Hartmanis and Richard Edwin Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[10] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[11] Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Texts in Computer Science. Springer, 4th edition, 2019.

[12] Michael Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2nd edition, 2006.

[13] The Coq Development Team. The Coq proof assistant, January 2022.

[14] René Thiemann and Lukas Schmidinger. The generalized multiset ordering is NP-complete. *Archive of Formal Proofs*, April 2022. https://isa-afp.org/entries/Multiset_Ordering_NPC.html, Formal proof development.

[15] Jian Xu, Xingyuan Zhang, Christian Urban, and Sebastiaan J. C. Joosten. Universal turing machine. *Archive of Formal Proofs*, February 2019. http://isa-afp.org/entries/Universal_Turing_Machine.html, Formal proof development.