A Formal Development of a Polychronous Polytimed Coordination Language

Hai Nguyen Van
hai.nguyenvan.phie@gmail.com

Frédéric Boulanger
frederic.boulanger@centralesupelec.fr

Burkhart Wolff
burkhart.wolff@lri.fr

July 31, 2019
## Contents

1 A Gentle Introduction to TESL
   1.1 Context ................................................. 5
   1.2 The TESL Language ....................................... 6
      1.2.1 Instantaneous Causal Operators ...................... 7
      1.2.2 Temporal Operators ................................ 7
      1.2.3 Asynchronous Operators ............................ 7

2 Core TESL: Syntax and Basics .................................... 9
   2.1 Syntactic Representation .................................. 9
      2.1.1 Basic elements of a specification ................... 9
      2.1.2 Operators for the TESL language ..................... 9
      2.1.3 Field Structure of the Metric Time Space .......... 10
   2.2 Defining Runs .............................................. 13

3 Denotational Semantics ........................................... 17
   3.1 Denotational interpretation for atomic TESL formulae .......... 17
   3.2 Denotational interpretation for TESL formulae ................ 18
      3.2.1 Image interpretation lemma ......................... 18
      3.2.2 Expansion law ...................................... 18
   3.3 Equational laws for the denotation of TESL formulae .......... 18
   3.4 Decreasing interpretation of TESL formulae ................ 19
   3.5 Some special cases ........................................ 21

4 Symbolic Primitives for Building Runs .......................... 23
   4.0.1 Symbolic Primitives for Runs ......................... 23
   4.1 Semantics of Primitive Constraints ........................ 24
      4.1.1 Defining a method for witness construction .......... 25
   4.2 Rules and properties of consistence ........................ 25
   4.3 Major Theorems .......................................... 26
      4.3.1 Interpretation of a context ......................... 26
      4.3.2 Expansion law ...................................... 26
   4.4 Equations for the interpretation of symbolic primitives .......... 26
      4.4.1 General laws ....................................... 26
      4.4.2 Decreasing interpretation of symbolic primitives .......... 27

5 Operational Semantics ........................................... 29
   5.1 Operational steps ......................................... 29
   5.2 Basic Lemmas ............................................. 31
# CONTENTS

6 **Semantics Equivalence**
   6.1 Stepwise denotational interpretation of TESL atoms .......................... 35
   6.2 Coinduction Unfolding Properties ............................................. 38
   6.3 Interpretation of configurations ............................................. 41

7 **Main Theorems**
   7.1 Initial configuration ............................................................... 47
   7.2 Soundness ................................................................................. 47
   7.3 Completeness ............................................................................. 50
   7.4 Progress .................................................................................... 52
   7.5 Local termination ................................................................. 60

8 **Properties of TESL**
   8.1 Stuttering Invariance ............................................................... 63
      8.1.1 Definition of stuttering ..................................................... 63
      8.1.2 Alternate definitions for counting ticks. ............................... 65
      8.1.3 Stuttering Lemmas ............................................................. 65
      8.1.4 Lemmas used to prove the invariance by stuttering ............... 66
      8.1.5 Main Theorems ................................................................. 86
Chapter 1

A Gentle Introduction to TESL

1.1 Context

The design of complex systems involves different formalisms for modeling their different parts or aspects. The global model of a system may therefore consist of a coordination of concurrent submodels that use different paradigms such as differential equations, state machines, synchronous data-flow networks, discrete event models and so on, as illustrated in Figure 1.1. This raises the interest in architectural composition languages that allow for “bolting the respective sub-models together”, along their various interfaces, and specifying the various ways of collaboration and coordination [2].

We are interested in languages that allow for specifying the timed coordination of subsystems by addressing the following conceptual issues:

- events may occur in different sub-systems at unrelated times, leading to polychronous systems, which do not necessarily have a common base clock,
- the behavior of the sub-systems is observed only at a series of discrete instants, and time coordination has to take this discretization into account,
- the instants at which a system is observed may be arbitrary and should not change its behavior (stuttering invariance),
- coordination between subsystems involves causality, so the occurrence of an event may enforce the occurrence of other events, possibly after a certain duration has elapsed or an event has occurred a given number of times,
- the domain of time (discrete, rational, continuous. . . ) may be different in the subsystems, leading to polytimed systems,
- the time frames of different sub-systems may be related (for instance, time in a GPS satellite and in a GPS receiver on Earth are related although they are not the same).

In order to tackle the heterogeneous nature of the subsystems, we abstract their behavior as clocks. Each clock models an event, i.e., something that can occur or not at a given time. This time is measured in a time frame associated with each clock, and the nature of time (integer, rational, real, or any type with a linear order) is specific to each clock. When the event associated
with a clock occurs, the clock ticks. In order to support any kind of behavior for the subsystems, we are only interested in specifying what we can observe at a series of discrete instants. There are two constraints on observations: a clock may tick only at an observation instant, and the time on any clock cannot decrease from an instant to the next one. However, it is always possible to add arbitrary observation instants, which allows for stuttering and modular composition of systems. As a consequence, the key concept of our setting is the notion of a clock-indexed Kripke model: \( \Sigma^\infty = \mathbb{N} \rightarrow \mathcal{K} \rightarrow (\mathbb{B} \times T) \), where \( \mathcal{K} \) is an enumerable set of clocks, \( \mathbb{B} \) is the set of booleans – used to indicate that a clock ticks at a given instant – and \( T \) is a universal metric time space for which we only assume that it is large enough to contain all individual time spaces of clocks and that it is ordered by some linear ordering \( (\leq_T) \).

The elements of \( \Sigma^\infty \) are called runs. A specification language is a set of operators that constrains the set of possible monotonic runs. Specifications are composed by intersecting the denoted run sets of constraint operators. Consequently, such specification languages do not limit the number of clocks used to model a system (as long as it is finite) and it is always possible to add clocks to a specification. Moreover, they are compositional by construction since the composition of specifications consists of the conjunction of their constraints.

This work provides the following contributions:

- defining the non-trivial language TESL* in terms of clock-indexed Kripke models,
- proving that this denotational semantics is stuttering invariant,
- defining an adapted form of symbolic primitives and presenting the set of operational semantic rules,
- presenting formal proofs for soundness, completeness, and progress of the latter.

### 1.2 The TESL Language

The TESL language [1] was initially designed to coordinate the execution of heterogeneous components during the simulation of a system. We define here a minimal kernel of operators that
1.2. THE TESL LANGUAGE

will form the basis of a family of specification languages, including the original TESL language, which is described at http://wdi.supelec.fr/software/TESL/.

1.2.1 Instantaneous Causal Operators

TESL has operators to deal with instantaneous causality, i.e., to react to an event occurrence in the very same observation instant.

- \( c_1 \) implies \( c_2 \) means that at any instant where \( c_1 \) ticks, \( c_2 \) has to tick too.
- \( c_1 \) implies not \( c_2 \) means that at any instant where \( c_1 \) ticks, \( c_2 \) cannot tick.
- \( c_1 \) kills \( c_2 \) means that at any instant where \( c_1 \) ticks, and at any future instant, \( c_2 \) cannot tick.

1.2.2 Temporal Operators

TESL also has chronometric temporal operators that deal with dates and chronometric delays.

- \( c \) sporadic \( t \) means that clock \( c \) must have a tick at time \( t \) on its own time scale.
- \( c_1 \) sporadic \( t \) on \( c_2 \) means that clock \( c_1 \) must have a tick at an instant where the time on \( c_2 \) is \( t \).
- \( c_1 \) time delayed by \( d \) on \( m \) implies \( c_2 \) means that every time clock \( c_1 \) ticks, \( c_2 \) must have a tick at the first instant where the time on \( m \) is \( d \) later than it was when \( c_1 \) had ticked. This means that every tick on \( c_1 \) is followed by a tick on \( c_2 \) after a delay \( d \) measured on the time scale of clock \( m \).
- time relation \((c_1, c_2)\) in \( R \) means that at every instant, the current time on clocks \( c_1 \) and \( c_2 \) must be in relation \( R \). By default, the time lines of different clocks are independent. This operator allows us to link two time lines, for instance to model the fact that time in a GPS satellite and time in a GPS receiver on Earth are not the same but are related. Time being polymorphic in TESL, this can also be used to model the fact that the angular position on the camshaft of an engine moves twice as fast as the angular position on the crankshaft \(^1\). We may consider only linear arithmetic relations to restrict the problem to a domain where the resolution is decidable.

1.2.3 Asynchronous Operators

The last category of TESL operators allows the specification of asynchronous relations between event occurrences. They do not specify the precise instants at which ticks have to occur, they only put bounds on the set of instants at which they should occur.

- \( c_1 \) weakly precedes \( c_2 \) means that for each tick on \( c_2 \), there must be at least one tick on \( c_1 \) at a previous or at the same instant. This can also be expressed by stating that at each instant, the number of ticks since the beginning of the run must be lower or equal on \( c_2 \) than on \( c_1 \).

\(^1\)See http://wdi.supelec.fr/software/TESL/GalleryEngine for more details
• \textit{c1 strictly precedes c2} means that for each tick on c2, there must be at least one tick on c1 at a previous instant. This can also be expressed by saying that at each instant, the number of ticks on c2 from the beginning of the run to this instant, must be lower or equal to the number of ticks on c1 from the beginning of the run to the previous instant.
Chapter 2

The Core of the TESL Language: Syntax and Basics

theory TESL
imports Main
begin

2.1 Syntactic Representation

We define here the syntax of TESL specifications.

2.1.1 Basic elements of a specification

The following items appear in specifications:

- Clocks, which are identified by a name.
- Tag constants are just constants of a type which denotes the metric time space.

datatype clock = Clk string
type synonym instant_index = nat
datatype 'τ tag_const = TConst (the_tag_const : 'τ) (τ_cst)

2.1.2 Operators for the TESL language

The type of atomic TESL constraints, which can be combined to form specifications.

datatype 'τ TESL_atomic =
  SporadicOn (clock) ('τ tag_const) (clock) (_ sporadic on _) 55
| TagRelation (clock) (clock) ('τ tag_const × 'τ tag_const) ⇒ bool
  ((time-relation [_, _] ∈ _) 55)
| Implies (clock) (clock)
  (infixr (implies) 55)
| ImpliesNot (clock) (clock)
  (infixr (implies not) 55)
| TimeDelayedBy (clock) ('τ tag_const) (clock) (clock)
  (_ time-delayed by _ on _ implies _) 55

9
CHAPTER 2. CORE TESL: SYNTAX AND BASICS

| WeaklyPrecedes (clock) (clock) | (infixr 'weakly precedes' 55) |
| StrictlyPrecedes (clock) (clock) | (infixr 'strictly precedes' 55) |
| Kills (clock) (clock) | (infixr 'kills' 55) |

A TESL formula is just a list of atomic constraints, with implicit conjunction for the semantics.

A TESL formula is just a list of atomic constraints, with implicit conjunction for the semantics.

type synonym 'τ TESL_formula = (τ TESL_atomic list)

We call positive atoms the atomic constraints that create ticks from nothing. Only sporadic constraints are positive in the current version of TESL.

fun positive_atom :: (τ TESL_atomic ⇒ bool) where
(positive_atom _ _ _ = True)
| (positive_atom _ _ _ = False)

The NoSporadic function removes sporadic constraints from a TESL formula.

abbreviation NoSporadic :: (τ TESL_formula ⇒ τ TESL_formula)
where
NoSporadic f ≡ (List.filter (λf_atom. case f_atom of
_ sporadic _ on _ ⇒ False
| _ ⇒ True) f))

2.1.3 Field Structure of the Metric Time Space

In order to handle tag relations and delays, tags must belong to a field. We show here that this is the case when the type parameter of τ tag_const is itself a field.

instantiation tag_const ::(field)field
begin
fun inverse_tag_const
where (inverse (τ cst t) = τ cst (inverse t))

fun divide_tag_const
where (divide (τ cst t1) (τ cst t2) = τ cst (divide t1 t2))

fun uminus_tag_const
where (uminus (τ cst t) = τ cst (uminus t))

fun minus_tag_const
where (minus (τ cst t1) (τ cst t2) = τ cst (minus t1 t2))

definition (one_tag Const ≡ τ cst 1)

fun times_tag_const
where (times (τ cst t1) (τ cst t2) = τ cst (times t1 t2))

definition (zero_tag Const ≡ τ cst 0)

fun plus_tag_const
where (plus (τ cst t1) (τ cst t2) = τ cst (plus t1 t2))

instance proof

fix a::(τ::field tag_const) and b::(τ::field tag_const)
and c::(τ::field tag_const)

obtain u v w where (a = τ cst u) and (b = τ cst v) and (c = τ cst w)
using tag_const.exhaust by metis

Multiplication is associative.
2.1. SYNTACTIC REPRESENTATION

thus \((a \cdot b \cdot c = a \cdot (b \cdot c))\)
by \((\text{simp add: TESL.times_tag_const.simps})\)

next

Multiplication is commutative.

fix \(a::('\tau::field tag_const)\) and \(b::('\tau::field tag_const)\)
obtain \(u\) \(v\) where \((a = \tau \text{cst } u)\) and \((b = \tau \text{cst } v)\) using \(\text{tag_const.exhaust}\) by \(\text{metis}\)
thus \((a \cdot b = b \cdot a)\)
by \((\text{simp add: TESL.times_tag_const.simps})\)

next

One is neutral for multiplication.

fix \(a::('\tau::field tag_const)\)
obtain \(u\) where \((a = \tau \text{cst } u)\) using \(\text{tag_const.exhaust}\) by \(\text{blast}\)
thus \((1 \cdot a = a)\)
by \((\text{simp add: TESL.times_tag_const.simps one_tag_const_def})\)

next

Addition is associative.

fix \(a::('\tau::field tag_const)\) and \(b::('\tau::field tag_const)\)
and \(c::('\tau::field tag_const)\)
obtain \(u\) \(v\) \(w\) where \((a = \tau \text{cst } u)\) and \((b = \tau \text{cst } v)\) and \((c = \tau \text{cst } w)\)
using \(\text{tag_const.exhaust}\) by \(\text{metis}\)
thus \((a + b + c = a + (b + c))\)
by \((\text{simp add: TESL.plus_tag_const.simps})\)

next

Addition is commutative.

fix \(a::('\tau::field tag_const)\) and \(b::('\tau::field tag_const)\)
obtain \(u\) \(v\) where \((a = \tau \text{cst } u)\) and \((b = \tau \text{cst } v)\) using \(\text{tag_const.exhaust}\) by \(\text{metis}\)
thus \((a + b = b + a)\)
by \((\text{simp add: TESL.plus_tag_const.simps})\)

next

Zero is neutral for addition.

fix \(a::('\tau::field tag_const)\)
obtain \(u\) where \((a = \tau \text{cst } u)\) using \(\text{tag_const.exhaust}\) by \(\text{blast}\)
thus \((0 + a = a)\)
by \((\text{simp add: TESL.plus_tag_const.simps zero_tag_const_def})\)

next

The sum of an element and its opposite is zero.

fix \(a::('\tau::field tag_const)\)
obtain \(u\) where \((a = \tau \text{cst } u)\) using \(\text{tag_const.exhaust}\) by \(\text{blast}\)
thus \((-a + a = 0)\)
by \((\text{simp add: TESL.plus_tag_const.simps TESL.uminus_tag_const.simps zero_tag_const_def})\)

next

Subtraction is adding the opposite.

fix \(a::('\tau::field tag_const)\) and \(b::('\tau::field tag_const)\)
obtain \(u\) \(v\) where \((a = \tau \text{cst } u)\) and \((b = \tau \text{cst } v)\) using \(\text{tag_const.exhaust}\) by \(\text{metis}\)
thus \((a - b = a + (-b))\)
by \((\text{simp add: TESL.minus_tag_const.simps})\)
Distributive property of multiplication over addition.

\[
\text{fix } a::('\tau::field tag_const) \text{ and } b::('\tau::field tag_const) \\
\text{and } c::('\tau::field tag_const) \\
\text{obtain } u v w \text{ where } (a = \tau\text{ cst } u) \text{ and } (b = \tau\text{ cst } v) \text{ and } (c = \tau\text{ cst } w) \\
\text{using } \text{tag_const.exhaust by metis} \\
\text{thus } ((a + b) * c = a * c + b * c) \text{ by (simp add: TESL.plus_tag_const.simps } \\
\text{TESL.times_tag_const.simps } \\
\text{ring_class.ring_distribs(2))}
\]

next

The neutral elements are distinct.

\[
\text{show } \langle (0::('\tau::field tag_const)) \neq 1 \rangle \text{ by (simp add: one_tag_const_def zero_tag_const_def)}
\]

next

The product of an element and its inverse is 1.

\[
\text{fix } a::('\tau::field tag_const) \text{ assume } h:(a \neq 0) \\
\text{obtain } u \text{ where } (a = \tau\text{ cst } u) \text{ using } \text{tag_const.exhaust by blast} \\
\text{moreover with } h \text{ have } (u \neq 0) \text{ by (simp add: zero_tag_const_def)} \\
\text{ultimately show } \langle \text{inverse } a * a = 1 \rangle \text{ by (simp add: TESL.inverse_tag_const.simps } \\
\text{TESL.times_tag_const.simps } \\
\text{one_tag_const_def)}
\]

next

Dividing is multiplying by the inverse.

\[
\text{fix } a::('\tau::field tag_const) \text{ and } b::('\tau::field tag_const) \\
\text{obtain } u v \text{ where } (a = \tau\text{ cst } u) \text{ and } (b = \tau\text{ cst } v) \text{ using } \text{tag_const.exhaust by metis} \\
\text{thus } (a \div b = a * \text{ inverse } b) \text{ by (simp add: TESL.divide_tag_const.simps } \\
\text{TESL.inverse_tag_const.simps } \\
\text{TESL.times_tag_const.simps } \\
\text{divide_inverse)}
\]

next

Zero is its own inverse.

\[
\text{show } \langle \text{inverse } (0::('\tau::field tag_const)) = 0 \rangle \text{ by (simp add: TESL.inverse_tag_const.simps zero_tag_const_def)}
\]

qed

end

For comparing dates (which are represented by tags) on clocks, we need an order on tags.

\[
\text{instantiation tag_const :: (order)order begin } \\
\text{inductive less_eq_tag_const :: ('a tag_const \Rightarrow 'a tag_const \Rightarrow bool) } \\
\text{where } \\
\text{Int_less_eq[simp]: } (n \leq m \Rightarrow (\text{TConst } n) \leq (\text{TConst } m)) \\
\text{definition less_tag: } ((x::'a tag_const) < y \longleftrightarrow (x \leq y) \land (x \neq y))
\]

end
instance proof
  show \(\forall x y :: 'a \tagconst. (x < y) = (x \leq y \land \neg y \leq x)\)
  using less_eq_tag_const.simps less_tag by auto
next
  fix \(x::'a \tagconst\)
  from tag_const.exhaust obtain \(x_0::'a\) where \(x = \TConst x_0\) by blast
  with Int.less_eq show \((x \leq x)\) by simp
next
  show \(\forall x y z :: 'a \tagconst. x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z\)
  using less_eq_tag_const.simps by auto
next
  show \(\forall x y :: 'a \tagconst. x \leq y \Longrightarrow y \leq x \Longrightarrow x = y\)
  using less_eq_tag_const.simps by auto
qed

end

For ensuring that time does never flow backwards, we need a total order on tags.

instantiation tag_const :: (linorder)linorder
begin
  instance proof
    fix \(x::'a \tagconst\) and \(y::'a \tagconst\)
    from tag_const.exhaust obtain \(x_0::'a\) where \(x = \TConst x_0\) by blast
    moreover from tag_const.exhaust obtain \(y_0::'a\) where \(y = \TConst y_0\) by blast
    ultimately show \((x \leq y \lor y \leq x)\) using less_eq_tag_const.simps by fastforce
qed
end

2.2 Defining Runs

theory Run
imports TESL
begin

Runs are sequences of instants, and each instant maps a clock to a pair \((h, t)\) where \(h\) indicates whether the clock ticks or not, and \(t\) is the current time on this clock. The first element of the pair is called the hamlet of the clock (to tick or not to tick), the second element is called the time.

abbreviation hamlet where \(\text{hamlet} \equiv \text{fst}\)
abbreviation time where \(\text{time} \equiv \text{snd}\)

type_synonym 'τ instant = ('clock ⇒ ('bool × 'τ tag_const))

Runs have the additional constraint that time cannot go backwards on any clock in the sequence of instants. Therefore, for any clock, the time projection of a run is monotonous.

typedef (overloaded) 'τ::linordered_field run = "
\{ ϱ::nat ⇒ 'τ instant. \forall c. mono (\lambda n. time (\rho n c)) \}
"
proof
  show \((\lambda _. \text{True}, \tau_{\text{est}} , 0)) \in \{ \rho. \forall c. mono (\lambda n. \text{time (\rho n c))}\);
  unfolding mono_def by blast
qed
CHAPTER 2. CORE TSL: SYNTAX AND BASICS

lemma Abs_run_inverse_rewrite:
\(\forall c. \text{mono} (\lambda n. \text{time} (\rho n c)) \Rightarrow \text{Rep}_\text{run} (\text{Abs}_\text{run} \rho) = \rho\)
by (simp add: Abs_run_inverse)

A dense run is a run in which something happens (at least one clock ticks) at every instant.

definition dense_run \(\rho\) \(\equiv\) \((\forall n. \exists c. \text{hamlet ((Rep}_\text{run} \rho) n c))\)

run_tick_count \(\rho\) \(K\) \(n\) counts the number of ticks on clock \(K\) in the interval \([0, n]\) of run \(\rho\).

fun run_tick_count :: \((\sigma::\text{linordered_field}) \Rightarrow \text{clock} \Rightarrow \text{nat} \Rightarrow \text{nat}\)
where \((\#_{\leq} \rho K 0) = (\text{if hamlet ((Rep}_\text{run} \rho) 0 K) then 1 else 0)\)
\((\#_{\leq} \rho K (\text{Suc} n)) = (\text{if hamlet ((Rep}_\text{run} \rho) (\text{Suc} n) K) then 1 + (\#_{\leq} \rho K n) else (\#_{\leq} \rho K n))\)

run_tick_count_strictly \(\rho\) \(K\) \(n\) counts the number of ticks on clock \(K\) in the interval \([0, n]\) of run \(\rho\).

fun run_tick_count_strictly :: \((\sigma::\text{linordered_field}) \Rightarrow \text{clock} \Rightarrow \text{nat} \Rightarrow \text{nat}\)
where \((\#_{<} \rho K 0) = 0\)
\((\#_{<} \rho K (\text{Suc} n)) = (\#_{\leq} \rho K n)\)

first_time \(\rho\) \(K\) \(n\) \(\tau\) tells whether instant \(n\) in run \(\rho\) is the first one where the time on clock \(K\) reaches \(\tau\).

definition first_time :: \((\sigma::\text{linordered_field}) \Rightarrow \text{clock} \Rightarrow \text{nat} \Rightarrow \text{tag_const} \Rightarrow \text{bool}\)
where \(\text{first}_\text{time} \rho K n \tau \equiv (\text{time ((Rep}_\text{run} \rho) n K) = \tau) \land (\exists n'. n' < n \land \text{time ((Rep}_\text{run} \rho) n' K) = \tau))\)

The time on a clock is necessarily less than \(\tau\) before the first instant at which it reaches \(\tau\).

lemma before_first_time:
assumes \(\text{first}_\text{time} \rho K n \tau\)
and \(m < n\)
shows \(\text{time ((Rep}_\text{run} \rho) m K) < \tau)\)
proof -
have \(\text{mono} (\lambda n. \text{time} (\text{Rep}_\text{run} \rho n K)))\) using \text{Rep}_\text{run} by blast
moreover from \text{assms}(2) have \(m \leq n\) using \text{less_imp_le} by simp
moreover have \(\text{mono} (\lambda n. \text{time} (\text{Rep}_\text{run} \rho n K)))\) using \text{Rep}_\text{run} by blast
ultimately have \(\text{time ((Rep}_\text{run} \rho) m K) \leq \text{time ((Rep}_\text{run} \rho) n K)})\)
by (simp add: mono_def)
moreover from \text{assms}(1) have \(\text{time ((Rep}_\text{run} \rho) n K) = \tau)\)
using \text{first_time_def} by blast
moreover from \text{assms} have \(\text{time ((Rep}_\text{run} \rho) m K) \neq \tau)\)
using \text{first_time_def} by blast
ultimately show \(?thesis by \text{simp}\)
qed

This leads to an alternate definition of \text{first_time}:

lemma alt_first_time_def:
assumes \(\forall m < n. \text{time ((Rep}_\text{run} \rho) m K) < \tau)\)
2.2. DEFINING RUNS

and \( \text{time } ((\text{Rep\_run } \varrho) \ n \ K) = \tau \) 
shows \( \text{first\_time } \varrho \ K \ n \ \tau \) 

proof - 
  from assms(1) have \( \forall n < n. \text{time } ((\text{Rep\_run } \varrho) \ n \ K) \neq \tau \) 
    by (simp add: less_le) 
  with assms(2) show ?thesis by (simp add: first_time_def) 
qed 

end
Chapter 3
Denotational Semantics

theory Denotational
imports
    TESL
    Run
begin

The denotational semantics maps TESL formulae to sets of satisfying runs. Firstly, we define the semantics of atomic formulae (basic constructs of the TESL language), then we define the semantics of compound formulae as the intersection of the semantics of their components: a run must satisfy all the individual formulae of a compound formula.

3.1 Denotational interpretation for atomic TESL formulae

fun TESL_interpretation_atomic :: ('τ::linordered_field) TESL_atomic ⇒ 'τ run set ([_] TESL)
where
  — K₁ sporadic τ on K₂ means that K₁ should tick at an instant where the time on K₂ is τ.
  ⟨[K₁ sporadic τ on K₂] TESL =
   (φ. ∃n::nat. hamlet ((Rep_run φ) n K₁) ∧ time ((Rep_run φ) n K₂) = τ)⟩
  — time-relation [K₁, K₂] ∈ R means that at each instant, the time on K₁ and the time on K₂ are in relation R.
  ⟨[time-relation [K₁, K₂] ∈ R] TESL =
   (φ. ∀n::nat. R (time ((Rep_run φ) n K₁), time ((Rep_run φ) n K₂)))⟩
  — master implies slave means that at each instant at which master ticks, slave also ticks.
  ⟨[master implies slave] TESL =
   (φ. ∀n::nat. hamlet ((Rep_run φ) n master) −→ hamlet ((Rep_run φ) n slave))⟩
  — master implies not slave means that at each instant at which master ticks, slave does not tick.
  ⟨[master implies not slave] TESL =
   (φ. ∀n::nat. hamlet ((Rep_run φ) n master) −→ ¬hamlet ((Rep_run φ) n slave))⟩
  — master time-delayed by δτ on measuring implies slave means that at each instant at which master ticks, slave will tick after a delay δτ measured on the time scale of measuring.
  ⟨[master time-delayed by δτ on measuring implies slave] TESL =
   (φ. ∀n::nat. hamlet ((Rep_run φ) n master) −→ hamlet ((Rep_run φ) n slave))⟩
   (let measured_time = time ((Rep_run φ) n measuring) in
    ∀m ≥ n. first_time measuring m (measured_time + δτ)⟩

17
— $K_1$ weakly precedes $K_2$ means that each tick on $K_2$ must be preceded by or coincide with at least one tick on $K_1$. Therefore, at each instant $n$, the number of ticks on $K_2$ must be less or equal to the number of ticks on $K_1$.

$\langle (\{ \varphi \mathrel{\#} \Phi \} \cap \varnothing) \rangle_{\text{TESL}} = \{ \varnothing \}$

— $K_1$ strictly precedes $K_2$ means that each tick on $K_2$ must be preceded by at least one tick on $K_1$ at a previous instant. Therefore, at each instant $n$, the number of ticks on $K_2$ must be less or equal to the number of ticks on $K_1$ at instant $n - 1$.

$\langle (\{ \varphi \mathrel{\#} \Phi \} \cap \varnothing) \rangle_{\text{TESL}} = \{ \varnothing \}$

— $K_1$ kills $K_2$ means that when $K_1$ ticks, $K_2$ cannot tick and is not allowed to tick at any further instant.

$\langle (\{ \varphi \mathrel{\#} \Phi \} \cap \varnothing) \rangle_{\text{TESL}} = \{ \varnothing \}$

### 3.2 Denotational interpretation for TESL formulae

To satisfy a formula, a run has to satisfy the conjunction of its atomic formulae. Therefore, the interpretation of a formula is the intersection of the interpretations of its components.

```plaintext
fun TESL_interpretation :: `(\\tau::linordered_field) TESL_formula \Rightarrow \tau run set
\langle [[[ \_ ]]_{\text{TESL}}] \rangle
where
\langle [[[ \_ ]]_{\text{TESL}}] = \{ \_, True \} \rangle
\langle [[[ \varphi \mathrel{\#} \Phi ]]_{\text{TESL}} = [[[ \varphi ]]_{\text{TESL}} \cap [[[ \Phi ]]_{\text{TESL}}] \rangle
```

#### 3.2.1 Image interpretation lemma

```plaintext
lemma TESL_interpretation_homo:
\langle [[[ \varphi ]]_{\text{TESL}} \cap [[[ \Phi ]]_{\text{TESL}} = [[[ \varphi \mathrel{\#} \Phi ]]_{\text{TESL}} \rangle
by simp
```

#### 3.2.2 Expansion law

Similar to the expansion laws of lattices.

```plaintext
theorem TESL_interpretation_image:
\langle [[[ \varphi ]]_{\text{TESL}} = \bigcap (\lambda \varphi'. [[[ \varphi' ]]_{\text{TESL}}] \cdot \text{set } \Phi) \rangle
by (induction \Phi, simp+)
```

#### 3.3 Equational laws for the denotation of TESL formulae

```plaintext
lemma TESL_interpret_assoc:
\langle [[[ \varphi_1 \mathrel{\#} \varphi_2 ] \mathrel{\#} \varphi_3 ]_{\text{TESL}} = [[[ \varphi_1 ]_{\text{TESL}} \cap [[[ \varphi_2 ]]_{\text{TESL}}] \rangle
by auto
```

```plaintext
lemma TESL_interpret_commute:
\langle [[[ \varphi_2 ]_{\text{TESL}} = [[[ \varphi_2 ]_{\text{TESL}} \cap [[[ \varphi_1 ]]_{\text{TESL}}] \rangle
by (simp add: TESL_interpret_homo_append inf_sup_aci(1))
```
3.4 Decreasing interpretation of TESL formulae

Adding constraints to a TESL formula reduces the number of satisfying runs.

 lemma TESL_sem_decreases_head:
  ⟨[[Φ]]TESL ⊇ [[[ϕ # Φ]]]TESL⟩
  by simp

 lemma TESL_sem_decreases_tail:
  ⟨[[Φ]]TESL ⊇ [[Φ @ [ϕ]]]TESL⟩
  by (simp add: TESL_interp_homo_append)

Repeating a formula in a specification does not change the specification.

 lemma TESL_interp_formula_stuttering:
  assumes ⟨ϕ ∈ set Φ⟩
  shows ⟨[[ϕ # Φ]]TESL = [[Φ]]TESL⟩
  proof -
    have ⟨ϕ # Φ = [ϕ] @ Φ⟩ by simp
    hence ⟨[[ϕ # Φ]]TESL = [[ϕ]]TESL ∩ [[Φ]]TESL⟩
      using TESL_interp_homo_append by simp
    thus ?thesis using assms TESL_interp_remdups_absorb
      by fastforce

Removing duplicate formulae in a specification does not change the specification.

 lemma TESL_interp_remdups_absorb:
  ⟨[[Φ]]TESL = [[rendups Φ]]TESL⟩
proof (induction "Φ")
  case Cons
  thus "case using TESL_interp_formula_stuttering by auto"
qed simp

Specifications that contain the same formulae have the same semantics.

lemma TESL_interp_set_lifting:
  assumes "set Φ ⊆ set Φ'"
  shows "[[ Φ ]] TESL ⊇ [[ Φ' ]] TESL"
proof -
  have "set (rdeps Φ) = set (rdeps Φ')"
    by (simp add: assms)
  moreover have "∀ r. (∃ ϕ. r ∈ rups Φ) → (∃ ϕ' ∈ rups Φ'. r ∈ rups Φ')"
    by (simp add: TESL_interp_remdups_absorb)
  moreover have "∀ r. (∃ ϕ. r ∈ rups Φ) → (∃ ϕ' ∈ rups Φ'. r ∈ rups Φ')"
    by (simp add: TESL_interp_remdups_absorb)
  ultimately show "thesis" using TESL_interp_rmdups_absorb by auto
qed

The semantics of specifications is contravariant with respect to their inclusion.

theorem TESL_interp_decreases_setinc:
  assumes "set Φ ⊆ set Φ'"
  shows "[[ Φ ]] TESL ⊇ [[ Φ' ]] TESL"
proof -
  obtain Φ, where decompose: "set (Φ Φ r) = set Φ'" using assms by auto
  hence "set (Φ Φ r) = set Φ'" using assms by blast
  moreover have "(set Φ) ∪ (set Φ r) = set Φ'"
    using assms decompose by auto
  moreover have "[[ Φ'] ] TESL = [[ Φ Φ r ]] TESL"
    using TESL_interp_set_lifting decompose by blast
  moreover have "[[ Φ Φ r ]] TESL = [[ Φ ]] TESL ∩ [[ Φ r ]] TESL"
    using TESL_interp_formula_stuttering[OF assms]
  ultimately show "thesis" by simp
qed

lemma TESL_interp_decreases_add_head:
  assumes "set Φ ⊆ set Φ'"
  shows "[[ ϕ # Φ ]] TESL ⊇ [[ ϕ # Φ' ]] TESL"
using assms TESL_interp_decreases_setinc by auto

lemma TESL_interp_decreases_add_tail:
  assumes "set Φ ⊆ set Φ'"
  shows "[[ Φ ∩ [ϕ] ]] TESL ⊇ [[ Φ' ∩ [ϕ] ]] TESL"
using TESL_interp_decreases_setinc[OF assms]
by (simp add: TESL_interp_formula_stuttering dual_order.trans)

lemma TESL_interp_absorb1:
  assumes "set Φ_1 ⊆ set Φ_2"
  shows "[[ Φ_1 ∩ Φ_2 ]] TESL = [[ Φ_2 ]] TESL"
by (simp add: Int_absorb1 TESL_interp_decreases_setinc TESL_interp_homo_append assms)

lemma TESL_interp_absorb2:
  assumes "set Φ_2 ⊆ set Φ_1"
  shows "[[ Φ_1 ∩ Φ_2 ]] TESL = [[ Φ_1 ]] TESL"
3.5 Some special cases

lemma NoSporadic_stable [simp]:
  ⟨[[Φ]]_{TESL} ⊆ [[NoSporadic Φ]]_{TESL}⟩
proof -
  from filter_is_subset have ⟨set (NoSporadic Φ) ⊆ set Φ⟩.
  from TESL_interp_decreases_setinc[OF this] show ?thesis.
  qed

lemma NoSporadic_idem [simp]:
  ⟨[[Φ]]_{TESL} ∩ [[NoSporadic Φ]]_{TESL} = [[Φ]]_{TESL}⟩
using NoSporadic_stable by blast

lemma NoSporadic_setinc:
  ⟨set (NoSporadic Φ) ⊆ set Φ⟩
by (rule filter_is_subset)

end
Chapter 4

Symbolic Primitives for Building Runs

theory SymbolicPrimitive
  imports Run

begin

We define here the primitive constraints on runs, towards which we translate TESL specifications
in the operational semantics. These constraints refer to a specific symbolic run and can therefore
access properties of the run at particular instants (for instance, the fact that a clock ticks at
instant \( n \) of the run, or the time on a given clock at that instant).

In the previous chapters, we had no reference to particular instants of a run because the TESL
language should be invariant by stuttering in order to allow the composition of specifications:
adding an instant where no clock ticks to a run that satisfies a formula should yield another run
that satisfies the same formula. However, when constructing runs that satisfy a formula, we need
to be able to refer to the time or hamlet of a clock at a given instant.

Counter expressions are used to get the number of ticks of a clock up to (strictly or not) a given
instant index.

datatype cnt_expr =
  TickCountLess {clock} {instant_index} {(#<)}
| TickCountLeq {clock} {instant_index} {(#≤)}

4.0.1 Symbolic Primitives for Runs

Tag values are used to refer to the time on a clock at a given instant index.

datatype tag_val =
  TSchemaic {clock * instant_index} {τvar}

datatype τ constr =
  — c ⇓ n @ τ constrains clock c to have time τ at instant n of the run.

  | Timestamp {clock} {instant_index} {τ tag_const} {(_⇓ _ @ _)}
    — m @ n ⊕ δt ⇒ n constrains clock s to tick at the first instant at which the time on m has increased by δt
      from the value it had at instant n of the run.
  | TimeDelay {clock} {instant_index} {τ tag_const} {clock} {(_⊕ _ ⇒ _)}
    — c ⊚ n constrains clock c to tick at instant n of the run.
The abstract machine has configurations composed of:

- the past $\Gamma$, which captures choices that have already been made as a list of symbolic primitive constraints on the run;
- the current index $n$, which is the index of the present instant;
- the present $\Psi$, which captures the formulae that must be satisfied in the current instant;
- the future $\Phi$, which captures the constraints on the future of the run.

**4.1 Semantics of Primitive Constraints**

The semantics of the primitive constraints is defined in a way similar to the semantics of TESL formulæ.

```plaintext
fun counter_expr_eval :: <!='r::linordered_field> run ⇒ cnt_expr ⇒ nat>
  (<!='r' cnt_expr>) ![cnt_expr]
where
  ![0 < clk indx ![cnt_expr] = run_tick_count_strictly 0 clk indx)
  ![0 ≤ clk indx ![cnt_expr] = run_tick_count 0 clk indx

fun symbolic_run_interpretation_primitive
  ::<!='r::linordered_field> constr ⇒ '!r run set) ([!] ![prim])
where
  ![K ▷ n ![prim] = (g. hamlet ((Rep_run g) n K))
  ![K ⊕ n0 ⊕ δt ⇒ K' ![prim] =
      (g. ∀n ≥ n0. first_time g K n (time ((Rep_run g) n0) δt) ⇒ hamlet ((Rep_run g) n K'))
  ![K ¬n < n ![prim] = (g. ¬hamlet ((Rep_run g) n K))
  ![K ¬n ≥ n ![prim] = (g. ∀i ≤ n. ¬ hamlet ((Rep_run g) i K))
  ![K n ⊕ τ ![prim] = (g. time ((Rep_run g) n K) ⇒ τ)
  ![τvar(K1, n1), τvar(K2, n2)] ∈ R ![prim] =
      { g. R (time ((Rep_run g) n1 K1), time ((Rep_run g) n2 K2)) }
4.2. RULES AND PROPERTIES OF CONSISTENCE

The composition of primitive constraints is their conjunction, and we get the set of satisfying runs by intersection.

```plaintext
fun symbolic_run_interpretation :: (α::linordered_field) constr list ⇒ (α::linordered_field) run set
where
  ⟨[[ ⊏ ]]prim = { ϱ. True }⟩

lemma symbolic_run_interp_cons_morph:
  ⟨[[ γ ]]prim ∩ [[ Γ ]]prim = [[ γ # Γ ]]prim⟩ by auto

definition consistent_context :: (α::linordered_field) constr list ⇒ bool
where
  ⟨consistent_context Γ ≡ ([[ Γ ]]prim ≠ {})⟩
```

4.1.1 Defining a method for witness construction

In order to build a run, we can start from an initial run in which no clock ticks and the time is always 0 on any clock.

```plaintext
abbreviation initial_run :: ((α::linordered_field) run) (⟨ϱ⟩) where
  ⟨ϱ⟩ ≡ Abs_run ((λ_ _. (False, α cst 0)) : nat ⇒ clock ⇒ (bool × 'α tag_const))
```

To help avoiding that time flows backward, setting the time on a clock at a given instant sets it for the future instants too.

```plaintext
fun time_update :: (nat ⇒ clock ⇒ ('α::linordered_field) tag_const ⇒ (nat ⇒ 'α instant) ⇒ (nat ⇒ 'α instant))
where
  ⟨time_update n k α ϱ = (λ n' k'. if k = k' ∧ n ≤ n' then (hamlet ϱ n k), α) else ϱ n' k')⟩
```

4.2 Rules and properties of consistence

```plaintext
lemma context_consistency_preservationI:
  (consistent_context ((γ::('α::linordered_field) constr)#Γ) ⇒ consistent_context Γ) unfolding consistent_context_def by auto
```

— This is very restrictive

```plaintext
inductive context_independency :: (α::linordered_field) constr ⇒ 'α constr list ⇒ bool (⊥ ≡ ⊥)
where
  NotTicks_independency:
  ⟨⟨K ⊏ n⟩/ α. n' ∈ set Γ ⇒ (K ⊏ n) ⊏ Γ⟩
| Ticks_independency:
  ⟨⟨K ⊏ n⟩/ α. n' ∈ set Γ ⇒ (K ⊏ n) ⊏ Γ⟩
| Timestamp_independency:
  ⟨⟨∀ τ'. τ' = τ ∧ (K ⊏ n ⊥) ∈ set Γ⟩ ⇒ (K ⊏ n ⊥ τ) ⊏ Γ⟩
```
4.3 Major Theorems

4.3.1 Interpretation of a context

The interpretation of a context is the intersection of the interpretation of its components.

\[
\bigcap (\lambda \gamma \cdot \llbracket \gamma \rrbracket_{\text{prim}} \cdot \text{set } \Gamma) = \llbracket \Gamma \rrbracket_{\text{prim}}
\]

by (induction \( \Gamma \), simp+)

4.3.2 Expansion law

Similar to the expansion laws of lattices

\[
\llbracket \llbracket \Gamma_1 \odot \Gamma_2 \rrbracket_{\text{prim}} = \llbracket \Gamma_1 \rrbracket_{\text{prim}} \cap \llbracket \Gamma_2 \rrbracket_{\text{prim}}
\]

by (induction \( \Gamma_1 \), simp, auto)

4.4 Equations for the interpretation of symbolic primitives

4.4.1 General laws

\[
\llbracket \llbracket (\Gamma_1 \odot \Gamma_2) \odot \Gamma_3 \rrbracket_{\text{prim}} = \llbracket \Gamma_1 \rrbracket_{\text{prim}} \cap \llbracket \Gamma_2 \rrbracket_{\text{prim}} \cap \llbracket \Gamma_3 \rrbracket_{\text{prim}}
\]

using symrun_interp_expansion by auto

\[
\llbracket \llbracket \Gamma_1 \odot \Gamma_2 \rrbracket_{\text{prim}} = \llbracket \Gamma_2 \odot \Gamma_1 \rrbracket_{\text{prim}}
\]

unfolding symrun_interp_expansion by auto

\[
\llbracket \llbracket \Gamma \odot \Gamma \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}}
\]

using symrun_interp_expansion by auto

\[
\llbracket \llbracket \Gamma_1 \odot (\Gamma_1 \odot \Gamma_2) \rrbracket_{\text{prim}} = \llbracket \Gamma_1 \odot \Gamma_2 \rrbracket_{\text{prim}}
\]

using symrun_interp_expansion by auto

\[
\llbracket \llbracket (\Gamma_1 \odot \Gamma_2) \odot \Gamma_2 \rrbracket_{\text{prim}} = \llbracket \Gamma_1 \odot \Gamma_2 \rrbracket_{\text{prim}}
\]

unfolding symrun_interp_expansion by auto

\[
\llbracket \llbracket \Box \odot \Gamma \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}}
\]

by simp

\[
\llbracket \llbracket \Gamma \odot \Box \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}}
\]

Identity element

\[
\llbracket \llbracket \Box \odot \Gamma \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}}
\]

by simp

\[
\llbracket \llbracket \Gamma \odot \Box \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}}
\]

by simp
4.4 EQUATIONS FOR THE INTERPRETATION OF SYMBOLIC PRIMITIVES

4.4.2 Decreasing interpretation of symbolic primitives

Adding constraints to a context reduces the number of satisfying runs.

\[
\text{lemma TESL\_sem\_decreases\_head:} \quad \langle \llbracket \Gamma \rrbracket_{\text{prim}} \supseteq \llbracket \gamma \# \Gamma \rrbracket_{\text{prim}} \rangle
\]

by simp

\[
\text{lemma TESL\_sem\_decreases\_tail:} \quad \langle \llbracket \Gamma \rrbracket_{\text{prim}} \supseteq \llbracket \Gamma \circ [\gamma] \rrbracket_{\text{prim}} \rangle
\]

by (simp add: symrun\_interp\_expansion)

Adding a constraint that is already in the context does not change the interpretation of the context.

\[
\text{lemma symrun\_interp\_formula\_stuttering:} \quad \text{assumes} \quad \langle \gamma \in \text{set } \Gamma \rangle \\
\text{shows} \quad \langle \llbracket \gamma \# \Gamma \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}} \rangle
\]

proof -
  have \(\gamma \# \Gamma = [\gamma] @ \Gamma\) by simp
  hence \(\langle \llbracket \gamma \# \Gamma \rrbracket_{\text{prim}} = \llbracket [\gamma] \rrbracket_{\text{prim}} \cap \llbracket \Gamma \rrbracket_{\text{prim}} \rangle\)
  using symrun\_interp\_expansion by simp
  thus \(?thesis using assms symrun\_interp\_fixpoint by fastforce\)
  qed

Removing duplicate constraints from a context does not change the interpretation of the context.

\[
\text{lemma symrun\_interp\_rendups\_absorb:} \quad \langle \llbracket \Gamma \rrbracket_{\text{prim}} = \llbracket \text{rendups } \Gamma \rrbracket_{\text{prim}} \rangle
\]

proof (induction \(\Gamma\))
  case Cons
  thus \(?case using symrun\_interp\_formula\_stuttering by auto\)
  qed simp

Two identical sets of constraints have the same interpretation, the order in the context does not matter.

\[
\text{lemma symrun\_interp\_set\_lifting:} \quad \text{assumes} \quad \langle \text{set } \Gamma = \text{set } \Gamma' \rangle \\
\text{shows} \quad \langle \llbracket \Gamma \rrbracket_{\text{prim}} = \llbracket \Gamma' \rrbracket_{\text{prim}} \rangle
\]

proof -
  have \(\text{set (rendups } \Gamma) = \text{set (rendups } \Gamma')\) by (simp add: assms)
  moreover have \(\text{fxpnt}\Gamma: \cap (\lambda\gamma. \llbracket \gamma \rrbracket_{\text{prim}} \circ \text{set } \Gamma) = \llbracket \Gamma \rrbracket_{\text{prim}}\)
  by (simp add: symrun\_interp\_fixpoint)
  moreover have \(\text{fxpnt}\Gamma': \cap (\lambda\gamma. \llbracket \gamma \rrbracket_{\text{prim}} \circ \text{set } \Gamma') = \llbracket \Gamma' \rrbracket_{\text{prim}}\)
  by (simp add: symrun\_interp\_fixpoint)
  moreover have \(\cap (\lambda\gamma. \llbracket \gamma \rrbracket_{\text{prim}} \circ \text{set } \Gamma) = \cap (\lambda\gamma. \llbracket \gamma \rrbracket_{\text{prim}} \circ \text{set } \Gamma')\)
  by (simp add: assms)
  ultimately show \(?thesis using symrun\_interp\_rendups\_absorb by auto\)
  qed

The interpretation of contexts is contravariant with regard to set inclusion.

\[
\text{theorem symrun\_interp\_decreases\_setinc:} \quad \text{assumes} \quad \langle \text{set } \Gamma \subseteq \text{set } \Gamma' \rangle \\
\text{shows} \quad \langle \llbracket \Gamma \rrbracket_{\text{prim}} \supseteq \llbracket \Gamma' \rrbracket_{\text{prim}} \rangle
\]

proof -
obtain \( \Gamma_r \) where decompose: \((\text{set} (\Gamma \oplus \Gamma_r) = \text{set} \Gamma')\) using asms by auto hence \((\text{set} (\Gamma \oplus \Gamma_r) = \text{set} \Gamma')\) using asms by blast moreover have \(((\text{set} \Gamma) \cup (\text{set} \Gamma_r) = \text{set} \Gamma')\) using asms decompose by auto moreover have \(\langle \text{set} (\Gamma \oplus \Gamma_r) = \text{set} \Gamma' \rangle\) using asms by auto moreover have \(\langle \text{set} (\Gamma) \cup \text{set} (\Gamma_r) = \text{set} \Gamma' \rangle\) using asms decompose by auto moreover have \(\langle [\Gamma] \subseteq [\Gamma'] \rangle\) by simp ultimately show ?thesis by simp qed

lemma symrun_interp_decreases_add_head:
  assumes \(\langle \text{set} \Gamma \subseteq \text{set} \Gamma' \rangle\)
  shows \(\langle [\gamma \# \Gamma] \supseteq [\gamma \# \Gamma'] \rangle\)
  using symrun_interp_decreases_setinc assms by auto

lemma symrun_interp_decreases_add_tail:
  assumes \(\langle \text{set} \Gamma \subseteq \text{set} \Gamma' \rangle\)
  shows \(\langle [\Gamma @ \gamma] \supseteq [\Gamma' @ \gamma] \rangle\)
  proof
    from symrun_interp_decreases_setinc[OF assms] have \(\langle [\Gamma'] \supseteq [\Gamma] \rangle\), thus ?thesis by (simp add: symrun_interp_expansion dual_order.trans)
  qed

lemma symrun_interp_absorb1:
  assumes \(\langle \text{set} \Gamma_1 \subseteq \text{set} \Gamma_2 \rangle\)
  shows \(\langle [\Gamma_1 @ \Gamma_2] = [\Gamma_2] \rangle\)
  by (simp add: Int_absorb1 symrun_interp_decreases_setinc symrun_interp_expansion assms)

lemma symrun_interp_absorb2:
  assumes \(\langle \text{set} \Gamma_1 \subseteq \text{set} \Gamma_2 \rangle\)
  shows \(\langle [\Gamma_1 @ \Gamma_2] = [\Gamma_2] \rangle\)
  using symrun_interp_absorb1 symrun_interp_commute assms by blast

end
Chapter 5

Operational Semantics

theory Operational
imports
SymbolicPrimitive

begin

The operational semantics defines rules to build symbolic runs from a TESL specification (a set of TESL formulae). Symbolic runs are described using the symbolic primitives presented in the previous chapter. Therefore, the operational semantics compiles a set of constraints on runs, as defined by the denotational semantics, into a set of symbolic constraints on the instants of the runs. Concrete runs can then be obtained by solving the constraints at each instant.

5.1 Operational steps

We introduce a notation to describe configurations:

- $\Gamma$ is the context, the set of symbolic constraints on past instants of the run;
- $n$ is the index of the current instant, the present;
- $\Psi$ is the TESL formula that must be satisfied at the current instant (present);
- $\Phi$ is the TESL formula that must be satisfied for the following instants (the future).

abbreviation uncurry_conf
:: ('\tau::linordered_field) system ⇒ instant_index ⇒ '\tau TESL_formula ⇒ '\tau TESL_formula
⇒ '\tau config
where
  ⟨\Gamma, n ⊢ \Psi \triangleright \Phi⟩ ≡ (\Gamma, n, \Psi, \Phi)

The only introduction rule allows us to progress to the next instant when there are no more constraints to satisfy for the present instant.

inductive operational_semantics_intro
:: ('\tau::linordered_field) config ⇒ '\tau config ⇒ bool
where
  instant_i:
The elimination rules describe how TESL formulae for the present are transformed into constraints on the past and on the future.

\[ (\Gamma, n \vdash [\square \tau \triangleright \Phi) \iff (\Gamma, \mathsf{Suc} \ n \vdash \Phi \triangleright [\square]) \]

inductive operational_semantics_elim

\[ ::= (\tau : \text{linordered_field}) \mathsf{config} \Rightarrow (\tau \mathsf{config} \Rightarrow \mathsf{bool}) \quad ((\_ \iff_e \_)) \]

where

\begin{itemize}
  \item sporadic_on_e1:
    A sporadic constraint can be ignored in the present and rejected into the future.
    \[ (\Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi) \]
    \[ \iff_e (\Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi)) \]
  \item sporadic_on_e2:
    It can also be handled in the present by making the clock tick and have the expected time. Once it has been handled, it is no longer a constraint to satisfy, so it disappears from the future.
    \[ (\Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi) \]
    \[ \iff_e (((K_1 \uparrow n) \# (K_2 \downarrow n \# \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi) \]
  \item weakly_precedes_e:
    A weak precedence relation has to be obeyed at every instant.
    \[ (\Gamma, n \vdash (K_1 \mathsf{ weakly precedes } K_2) \# \Psi) \triangleright \Phi) \]
    \[ \iff_e (((K_1 \mathsf{ weakly precedes } K_2) \# \Psi) \triangleright \Phi) \]
  \item strictly_precedes_e:
    A strict precedence relation has to hold at every instant.
    \[ (\Gamma, n \vdash (K_1 \mathsf{ strictly precedes } K_2) \# \Psi) \triangleright \Phi) \]
    \[ \iff_e (((K_1 \mathsf{ strictly precedes } K_2) \# \Psi) \triangleright \Phi) \]
\end{itemize}
5.2. BASIC LEMMAS

\[(\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi) \quad \leftrightarrow_e \quad (((\equiv \geq K_2 n, \equiv K_1 n) \in (\lambda(x, y). x \leq y) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi))\]

<table>
<thead>
<tr>
<th>kills_el:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A kill can be handled by forbidding a tick of the triggering clock.</td>
</tr>
</tbody>
</table>

\[(\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi) \quad \leftrightarrow_e \quad (((K_1 \nless n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi))\]

<table>
<thead>
<tr>
<th>kills_e2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>It can also be handled by making the triggering clock tick and by forbidding any further tick of the killed clock.</td>
</tr>
</tbody>
</table>

\[(\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi) \quad \leftrightarrow_e \quad (((K_1 \nless n) \# (K_2 \nless K_1 \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi))\]

A step of the operational semantics is either the application of the introduction rule or the application of an elimination rule.

inductive operational_semantics_step
:::("τ::linordered_field") config ⇒ "τ" config ⇒ bool

where

intro_part:
\[(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \quad \leftrightarrow_i \quad (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) \quad \Rightarrow (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \quad \leftrightarrow (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)\]

elsims_part:
\[(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \quad \leftrightarrow_e \quad (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) \quad \Rightarrow (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \quad \leftrightarrow (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)\]

\[
\begin{align*}
\text{abbreviation } & \text{ operational_semantics_step_rtranclp } \quad := (\tau::\text{linordered_field}) \text{ config } \Rightarrow \tau \text{ config } \Rightarrow \text{ bool} \quad (\_ \leftrightarrow \_ \_ 70) \\
\text{ where } & \quad C_1 \equiv \tau::\text{linordered_field} \quad C_2 \equiv \text{ operational_semantics_step_rtranclp} \quad 70 \\
\text{abbreviation } & \text{ operational_semantics_step_tranclp } \quad := (\tau::\text{linordered_field}) \text{ config } \Rightarrow \tau \text{ config } \Rightarrow \text{ bool} \quad (\_ \leftrightarrow \_ \_ 70) \\
\text{ where } & \quad C_1 \equiv \tau::\text{linordered_field} \quad C_2 \equiv \text{ operational_semantics_step_tranclp} \quad 70 \\
\text{abbreviation } & \text{ operational_semantics_step_reflclp } \quad := (\tau::\text{linordered_field}) \text{ config } \Rightarrow \tau \text{ config } \Rightarrow \text{ bool} \quad (\_ \leftrightarrow \_ \_ 70) \\
\text{ where } & \quad C_1 \equiv \tau::\text{linordered_field} \quad C_2 \equiv \text{ operational_semantics_step_reflclp} \quad 70 \\
\text{abbreviation } & \text{ operational_semantics_step_relpowp } \quad := (\tau::\text{linordered_field}) \text{ config } \Rightarrow \text{ nat } \Rightarrow \tau \text{ config } \Rightarrow \text{ bool} \quad (\_ \leftrightarrow \_ \_ 70) \\
\text{ where } & \quad C_1 \equiv \tau::\text{linordered_field} \quad C_2 \equiv \text{ operational_semantics_step_relpowp} \quad 70 \\
\text{definition } & \text{ operational_semantics_elim_inv } \quad := (\tau::\text{linordered_field}) \text{ config } \Rightarrow \tau \text{ config } \Rightarrow \text{ bool} \quad (\_ \leftrightarrow \_ \_ 70) \\
\text{ where } & \quad C_1 \equiv \tau::\text{linordered_field} \quad C_2 \equiv \text{ operational_semantics_elim_inv} \quad 70 \\
\end{align*}
\]

5.2 Basic Lemmas

If a configuration can be reached in \( m \) steps from a configuration that can be reached in \( n \) steps from an original configuration, then it can be reached in \( n + m \) steps from the original.
configuration.

lemma operational_semantics_trans_generalized:
  assumes \( \langle C_1 \rightarrow^n C_2 \rangle \)
  assumes \( \langle C_2 \rightarrow^* C_3 \rangle \)
  shows \( \langle C_1 \rightarrow^* C_3 \rangle \)
using relcomp.relcompI[of operational_semantics_step ^^ n] .
by (simp add: operational_semantics_step.simps)

We consider the set of configurations that can be reached in one operational step from a given configuration.

abbreviation Cnext_solve
  :: ('\tau::linordered_field) config \Rightarrow '\tau config set \( \langle \mathcal{C}_{next} \_\rangle \)
where
\( \mathcal{C}_{next} \_ \equiv \{ \mathcal{S}', \mathcal{S} \rightarrow \mathcal{S}' \} \)

Advancing to the next instant is possible when there are no more constraints on the current instant.

lemma Cnext_solve_instant:
  \( \langle \mathcal{C}_{next} \_ (\Gamma, n \vdash \{} \vdash \Phi) \rangle \supseteq \{ \Gamma, \Suc n \vdash \{} \vdash \Phi \} \)
by (simp add: operational_semantics_step.simps operational_semantics_intro.instant_i)

The following lemmas state that the configurations produced by the elimination rules of the operational semantics belong to the configurations that can be reached in one step.

lemma Cnext_solve_sporadicon:
  \( \langle \mathcal{C}_{next} \_ (\Gamma, n \vdash (K_i \text{ sporadic } \tau \text{ on } K_j) \# \Psi) \vdash \Phi) \rangle \supseteq \{ \Gamma, n \vdash \Psi \triangleright ((K_i \text{ sporadic } \tau \text{ on } K_j) \# \Phi), \( ((K_i \vdash n) \# (K_j \vdash n \# \tau) \# \Phi), n \vdash \Psi \triangleright \Phi \} \)
by (simp add: operational_semantics_step.simps operational_semantics_elim.sporadic_on_el operational_semantics_elim.sporadic_on_e2)

lemma Cnext_solve_tagrel:
  \( \langle \mathcal{C}_{next} \_ (\Gamma, n \vdash (\text{time-relation } [K_1, K_2] \in R) \# \Psi) \vdash \Phi) \rangle \supseteq \{ \langle \tau_{var}(K_1, n), \tau_{var}(K_2, n) \in R \# \Gamma \rangle, \tau_{\Psi} \triangleright ((\text{time-relation } [K_1, K_2] \in R) \# \Phi) \}
by (simp add: operational_semantics_step.simps operational_semantics_elim.tagrel_e)

lemma Cnext_solve_implies:
  \( \langle \mathcal{C}_{next} \_ (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \vdash \Phi) \rangle \supseteq \{ \langle K_1 \dashv \triangleright n \# \Gamma \rangle, n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi), ((K_1 \vdash \triangleright n) \# (K_2 \vdash \triangleright n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \}
by (simp add: operational_semantics_step.simps operational_semantics_elim.implies_el operational_semantics_elim.implies_e2)

lemma Cnext_solve_implies_not:
  \( \langle \mathcal{C}_{next} \_ (\Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \vdash \Phi) \rangle \supseteq \{ \langle K_1 \dashv \triangleright n \# \Gamma \rangle, n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi), ((K_1 \vdash \triangleright n) \# (K_2 \vdash \triangleright n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \}
by (simp add: operational_semantics_step.simps operational_semantics_elim.implies_not_el operational_semantics_elim.implies_not_e2)

lemma Cnext_solve_timedelayed:
  \( \langle \mathcal{C}_{next} \_ (\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \vdash \Phi) \rangle \supseteq \{ \langle K_1 \dashv \triangleright n \# \Gamma \rangle, n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \}

5.2. BASIC LEMMAS

\[(K_1 \uparrow n) \# (K_2 \ominus n \oplus \delta_r \Rightarrow K_3) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta_r \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] 

by (simp add: operational_semantics_step.simps operational_semantics_elim.timedelayed_e1 operational_semantics_elim.timedelayed_e2)

lemma Cnext_solve_weakly_precedes:
\[\langle (C_{next} (\Gamma, n \vdash (K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi) \rangle \supseteq \langle \langle [\leq K_1 n, \leq K_2 n] \in (\lambda (x,y). x \leq y) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \rangle \] 

by (simp add: operational_semantics_step.simps operational_semantics_elim.weakly_precedes_e)

lemma Cnext_solve_strictly_precedes:
\[\langle (C_{next} (\Gamma, n \vdash (K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi) \rangle \supseteq \langle \langle [\leq K_1 n, < K_2 n] \in (\lambda (x,y). x \leq y) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi) \rangle \] 

by (simp add: operational_semantics_step.simps operational_semantics_elim.strictly_precedes_e)

lemma Cnext_solve_kills:
\[\langle (C_{next} (\Gamma, n \vdash (K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi) \rangle \supseteq \langle \langle \neg \vec{\downarrow} K_1 n \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi), (K_1 \uparrow n) \# (K_2 \neg \vec{\downarrow} n \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rangle \] 

by (simp add: operational_semantics_step.simps operational_semantics_elim.kills_e1 operational_semantics_elim.kills_e2)

An empty specification can be reduced to an empty specification for an arbitrary number of steps.

lemma empty_spec_reductions:
\[\langle [], 0 \vdash [] \triangleright [] \rangle \Rightarrow^* (k, k \vdash [] \triangleright []) \]

proof (induct k)
  case 0 thus ?case by simp
next
  case Suc thus ?case
    using instant_i operational_semantics_step.simps by fastforce
qed

end
Chapter 6

Equivalence of the Operational and Denotational Semantics

theory Corecursive_Prop
imports
  SymbolicPrimitive
  Operational
  Denotational
begin

6.1 Stepwise denotational interpretation of TESL atoms

In order to prove the equivalence of the denotational and operational semantics, we need to be able to ignore the past (for which the constraints are encoded in the context) and consider only the satisfaction of the constraints from a given instant index. For this purpose, we define an interpretation of TESL formulae for a suffix of a run. That interpretation is closely related to the denotational semantics as defined in the preceding chapters.

fun TESL_interpretation_atomic_stepwise :: ⟨('τ::linordered_field) TESL_atomic ⇒ nat ⇒ 'τ run set) (⋯)⟩

where

⟨(⋯)⟩

| ⟨[⋯]⟩ TESL ≥ i = {ϱ. ∃n ≥ i. hamlet ((Rep_run ϱ n K₁) ∨ time ((Rep_run ϱ n K₂) = τ)}
| ⟨[⋯]⟩ time-relation [K₁, K₂] ∈ R TESL ≥ i = {ϱ. ∀n ≥ i. R (time ((Rep_run ϱ n K₁), time ((Rep_run ϱ n K₂)))
| ⟨[⋯]⟩ master implies slave TESL ≥ i = {ϱ. ∀n ≥ i. hamlet ((Rep_run ϱ n master) → hamlet ((Rep_run ϱ n slave))
| ⟨[⋯]⟩ master implies not slave TESL ≥ i = {ϱ. ∀n ≥ i. hamlet ((Rep_run ϱ n master) → ¬ hamlet ((Rep_run ϱ n slave))
| ⟨[⋯]⟩ master time-delayed by δτ on measuring implies slave TESL ≥ i = {ϱ. ∀n ≥ i. hamlet ((Rep_run ϱ n master) → let measured_time = time ((Rep_run ϱ n measuring) in
  ∀m ≥ n. first_time ϱ measuring m (measured_time + δτ) → hamlet ((Rep_run ϱ n slave))

| ⟨[⋯]⟩ K₁ weakly precedes K₂ TESL ≥ i = {ϱ. ∀n ≥ i. (run_tick_count ϱ K₂ n) ≤ (run_tick_count ϱ K₁ n)}

35
The denotational interpretation of TESL formulae can be unfolded into the stepwise interpretation.

Lemma TESL_interp_unfold_stepwise_sporadicon:
\[
\langle [\ [K_1 \text{ sporadic } \tau \text{ on } K_2 ] ]_{TESL} = \bigcup \{ Y. \ \exists n::\text{nat}. Y = [ [K_1 \text{ sporadic } \tau \text{ on } K_2 ] ]_{TESL} \geq n \} \rangle
\]
by auto

Lemma TESL_interp_unfold_stepwise_tagrelgen:
\[
\langle [ [\ \text{time-relation} \ [K_1, K_2] ] ]_{TESL} = \bigcap \{ Y. \ \exists n::\text{nat}. Y = [ [\ \text{time-relation} \ [K_1, K_2] ] ]_{TESL} \geq n \} \rangle
\]
by auto

Lemma TESL_interp_unfold_stepwise_implies:
\[
\langle [ [\ \text{master implies slave} ] ]_{TESL} = \bigcap \{ Y. \ \exists n::\text{nat}. Y = [ [\ \text{master implies slave} ] ]_{TESL} \geq n \} \rangle
\]
by auto

Lemma TESL_interp_unfold_stepwise_implies_not:
\[
\langle [ [\ \text{master implies not slave} ] ]_{TESL} = \bigcap \{ Y. \ \exists n::\text{nat}. Y = [ [\ \text{master implies not slave} ] ]_{TESL} \geq n \} \rangle
\]
by auto

Lemma TESL_interp_unfold_stepwise_timedelayed:
\[
\langle [ [\ \text{master time-delayed by } \delta \tau \text{ on measuring implies slave} ] ]_{TESL} = \bigcap \{ Y. \ \exists n::\text{nat}. Y = [ [\ \text{master time-delayed by } \delta \tau \text{ on measuring implies slave} ] ]_{TESL} \geq n \} \rangle
\]
by auto

Lemma TESL_interp_unfold_stepwise_weakly_precedes:
\[
\langle [ [K_1 \text{ weakly precedes } K_2 ] ]_{TESL} = \bigcap \{ Y. \ \exists n::\text{nat}. Y = [ [K_1 \text{ weakly precedes } K_2 ] ]_{TESL} \geq n \} \rangle
\]
by auto

Lemma TESL_interp_unfold_stepwise_strictly_precedes:
\[
\langle [ [K_1 \text{ strictly precedes } K_2 ] ]_{TESL} = \bigcap \{ Y. \ \exists n::\text{nat}. Y = [ [K_1 \text{ strictly precedes } K_2 ] ]_{TESL} \geq n \} \rangle
\]
by auto

Lemma TESL_interp_unfold_stepwise_kills:
\[
\langle [ [\ \text{master kills slave} ] ]_{TESL} = \bigcap \{ Y. \ \exists n::\text{nat}. Y = [ [\ \text{master kills slave} ] ]_{TESL} \geq n \} \rangle
\]
by auto

Positive atomic formulae (the ones that create ticks from nothing) are unfolded as the union of the stepwise interpretations.

Theorem TESL_interp_unfold_stepwise_positive_atoms:

Assumes positive_atom \( \phi \)

Shows \( [\ [\ \phi ::'\tau ::\text{linordered_field TESL}\_\text{atomic} \ ] ]_{TESL} = \bigcup \{ Y. \ \exists n::\text{nat}. Y = [ [\ \phi ] ]_{TESL} \geq n \} \)

Proof -

From positive_atom.elims(2)[OF assms]

Obtain u v w where \( \phi = (u \text{ sporadic } v \text{ on } w) \) by blast

With TESL_interp_unfold_stepwise_sporadicon show ?thesis by simp qed
Negative atomic formulae are unfolded as the intersection of the stepwise interpretations.

**Theorem TESL_interp_unfold_stepwise_negative_atoms:**

assumes \( \neg \text{positive_atom } \varphi \)

shows \( \langle \big[ \big[ \varphi \big] \rangle_{TESL} = \bigcap \{ \exists n :: \text{nat}. \ Y = \big[ \big[ \varphi \big] \big]_{TESL} \geq n \} \rangle \)

proof (cases \( \varphi \))

case SporadicOn

thus ?thesis using \( \text{assms} \) by simp

case TagRelation

thus ?thesis using TESL_interp_unfold_stepwise_tagrelgen by simp

case Implies

thus ?thesis using TESL_interp_unfold_stepwise_implies by simp

case ImpliesNot

thus ?thesis using TESL_interp_unfold_stepwise_implies_not by simp

case TimeDelayedBy

thus ?thesis using TESL_interp_unfold_stepwise_timedelayed by simp

case WeaklyPrecedes

thus ?thesis using TESL_interp_unfold_stepwise_weakly_precedes by simp

case StrictlyPrecedes

thus ?thesis using TESL_interp_unfold_stepwise_strictly_precedes by simp

case Kills

thus ?thesis using TESL_interp_unfold_stepwise_kills by simp

case KillNot

thus ?thesis using TESL_interp_unfold_stepwise_kills_not by simp

qed

Some useful lemmas for reasoning on properties of sequences.

**Lemma forall_nat_expansion:**

\( \langle \forall n \geq (n_0 :: \text{nat}). \ P n \rangle = (P n_0 \land (\forall n \geq n_0. \ P n)) \)

proof

have \( \langle \forall n \geq (n_0 :: \text{nat}). \ P n \rangle = (\forall n. \ (n = n_0 \lor n > n_0) \rightarrow P n) \)

using le_less by blast

also have \( \ldots = (P n_0 \land (\forall n > n_0. \ P n)) \) by blast

finally show ?thesis using Suc_le_eq by simp

qed

**Lemma exists_nat_expansion:**

\( \langle \exists n \geq (n_0 :: \text{nat}). \ P n \rangle = (P n_0 \lor (\exists n \geq n_0. \ P n)) \)

proof

have \( \langle \exists n \geq (n_0 :: \text{nat}). \ P n \rangle = (\exists n. \ (n = n_0 \lor n > n_0) \land P n) \)

using le_less by blast

also have \( \ldots = (\exists n. \ (P n_0) \lor (n > n_0 \land P n)) \) by blast

finally show ?thesis using Suc_le_eq by simp

qed

**Lemma forall_nat_set_suc:**

\( \langle \forall m \geq n. \ P \ x \ m \rangle = (x. \ P \ x \ n) \cap (x. \ \forall m \geq n_0. \ P \ x \ m) \)

proof

\{ fix x assume h: x \in (x. \ \forall m \geq n. \ P \ x \ n) \\
  hence P x n by simp \\
  moreover from h have \( x \in (x. \ \forall m \geq n_0. \ P \ x \ m) \) by simp \\
  ultimately have \( x \in (x. \ P \ x \ n) \cap (x. \ \forall m \geq n_0. \ P \ x \ m) \) by simp \\
  \} thus \( (x. \ \forall m \geq n. \ P \ x \ m) \subseteq (x. \ P \ x \ n) \cap (x. \ \forall m \geq n_0. \ P \ x \ m) \) ..
next \{ \text{fix } x \text{ assume } h':(x \in (x. \ P x n) \cap (x. \ \forall m \geq \text{Suc } n. \ P x m)) \\
\text{hence } (P x n) \text{ by simp} \\
\text{moreover from } h \text{ have } (\forall n \geq \text{Suc } n. \ P x m) \text{ by simp} \\
\text{ultimately have } (\forall n \geq \text{n}. \ P x m) \text{ using forall_nat_expansion by blast} \\
\text{hence } (x \in (x. \ \forall m \geq \text{n}. \ P x m)) \text{ by simp} \} \text{ thus } (x \in (x. \ \forall m \geq \text{n}. \ P x m)) \subseteq (x. \ \forall m \geq \text{n}. \ P x m).
\text{.. qed}

lemma exists_nat_set_suc: (x. \exists n \geq \text{n}. \ P x m) = (x. \ P x n) \cup (x. \ \exists m \geq \text{Suc } n. \ P x m)
proof \{ \text{fix } x \text{ assume } h:(x \in (x. \ \exists n \geq \text{n}. \ P x m)) \\
\text{hence } (x \in (x. \ \exists n \geq \text{n}. \ P x m)) \text{ by simp} \\
\text{ultimately have } (\forall n \geq \text{n}. \ P x m) \text{ using Suc_le_eq antisym_conv2 by fastforce} \\
\text{hence } (x \in (x. \ \exists n \geq \text{n}. \ P x m)) \text{ by blast} \} \text{ thus } (x. \ \exists n \geq \text{n}. \ P x m) \subseteq (x. \ \exists m \geq \text{n}. \ P x m).
\text{.. qed}

\section{Coinduction Unfolding Properties}

The following lemmas show how to shorten a suffix, i.e. to unfold one instant in the construction of a run. They correspond to the rules of the operational semantics.

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{proof}

\textbf{by (simp add: Collect_conj_eq)}

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

6.2 Coinduction Unfolding Properties

The following lemmas show how to shorten a suffix, i.e. to unfold one instant in the construction of a run. They correspond to the rules of the operational semantics.

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)

\textbf{lemma TESL_interpretation_atomic_stepwise.simps(1)}

symbolic_run_interpretation_primitive.simps(1, 6)

\textbf{using exists_nat_set_suc[of n] (λ x h. \text{hamlet} (Rep_run x n K1)}

\textbf{and time (Rep_run x n K2) = τ) by simp add: Collect_conj_eq)
proof -
have \((\varphi. \forall m \geq n. \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{slave}))\)
= \((\varphi. \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{slave}))\)
\(\cap \{\varphi. \forall m \geq \text{Suc} n. \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{slave})\}\)
using \(\forall\text{nat_set suc[of } n \backslash \lambda x y. \text{hamlet} ((\text{Rep}_\varphi x) y \text{ master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi x) y \text{ slave})\}\) by simp
thus \(?thesis\) by auto
qed

lemma \(\text{TESL interp stepwise_implies_not_coind_unfold}:\)
\[
\begin{align*}
\{ \text{master implies not slave} \}_{\text{TESL} \geq n} & = \\
\{ \text{master \neg \parallel n} \}_{\text{prim}} & \quad \text{— rule implies_not_e1} \\
\cup \{ \text{master \parallel n} \}_{\text{prim}} \cap \{ \text{slave \neg \parallel n} \}_{\text{prim}} & \quad \text{— rule implies_not_e2} \\
\cap \{ \text{master implies not slave} \}_{\text{TESL} \geq \text{Suc} n} & \quad \text{— rule implies_not_e3}
\end{align*}
\]
proof -
have \((\varphi. \forall m \geq n. \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{slave}))\)
= \((\varphi. \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{slave}))\)
\(\cap \{\varphi. \forall m \geq \text{Suc} n. \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{slave})\}\)
using \(\forall\text{nat_set suc[of } n \backslash \lambda x y. \text{hamlet} ((\text{Rep}_\varphi x) y \text{ master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi x) y \text{ slave})\}\) by simp
thus \(?thesis\) by auto
qed

lemma \(\text{TESL interp stepwise_timedelayed_coind_unfold}:\)
\[
\begin{align*}
\{ \text{master time-delayed by } \delta \tau \text{ on measuring implies slave} \}_{\text{TESL} \geq n} & = \\
\{ \text{master \neg \parallel n} \}_{\text{prim}} & \quad \text{— rule timedelayed_e1} \\
\cup \{ \text{master \parallel n} \}_{\text{prim}} \cap \{ \text{measuring \parallel n} \} & \quad \text{— rule timedelayed_e2} \\
\cap \{ \text{master time-delayed by } \delta \tau \text{ on measuring implies slave} \}_{\text{TESL} \geq \text{Suc} n} & \quad \text{— rule timedelayed_e3}
\end{align*}
\]
proof -
let \(?prop = \langle \lambda \varphi m. \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{master} \rangle \rightarrow \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{slave})\)
(let \(\text{measured_time} = \text{time} ((\text{Rep}_\varphi m) \rightarrow \text{measuring})\) in
\(\forall p \geq m. \text{first_time} \varphi \text{ measuring p (measured_time + } \delta \tau) \rightarrow \text{hamlet} ((\text{Rep}_\varphi m) \rightarrow \text{slave})\))
have \((\varphi. \forall m \geq n. \text{?prop } m) = \{\varphi. \text{?prop } m\} \cap \{\varphi. \forall m \geq \text{Suc} n. \text{?prop } m\}\)
using \(\forall\text{nat_set suc[of } n \backslash \text{?prop} \) by blast
also have \(\cdots = \{\varphi. \text{?prop } m\}\)
\(\cap \{ \text{master time-delayed by } \delta \tau \text{ on measuring implies slave} \}_{\text{TESL} \geq \text{Suc} n}\)
by simp
finally show \(?thesis\) by auto
qed

lemma \(\text{TESL interp stepwise_weakly_precedes_coind_unfold}:\)
\[
\begin{align*}
\{ \text{K1 weakly precedes } K2 \}_{\text{TESL} \geq n} & = \\
\{ \langle \# \leq K2 n, \# \leq K1 n \rangle \in (\lambda (x,y). x \leq y) \}_{\text{prim}} & \quad \text{— rule weakly_precedes_e} \\
\cap \{ \text{K1 weakly precedes } K2 \}_{\text{TESL} \geq \text{Suc} n} & \quad \text{— rule weakly_precedes_e}
\end{align*}
\]
proof -
have \((\varphi. \forall p \geq n. \text{run_tick_count } p \leq \text{run_tick_count } K2 p) \leq \text{run_tick_count } K1 p)\)
= \((\varphi. \text{run_tick_count } p \leq \text{run_tick_count } K2 p) \leq \text{run_tick_count } K1 p)\)
\(\cap \{\varphi. \forall p \geq \text{Suc} n. \text{run_tick_count } p \leq \text{run_tick_count } K2 p) \leq \text{run_tick_count } K1 p)\)
using \(\forall\text{nat_set suc[of } n \backslash \lambda \varphi n. \text{run_tick_count } p \leq \text{run_tick_count } K2 p\)\)
\(\leq \text{run_tick_count } K1 p)\)
by simp
thus \(?thesis\) by auto
qed
lemma TESL_interp_stepwise_strictly_precedes_coind_unfold:

\[
\begin{align*}
\{ K_1 \text{ strictly precedes } K_2 \} & \text{TESL}^{\geq n} = \\
\bigl( \{ \sigma \leq K_2, K_2 \not\subset K_1 \} \bigr) & \text{prim} \\
\bigcap & \bigl( K_1 \text{ strictly precedes } K_2 \} & \text{TESL}^{\geq \text{Suc } n}\bigr) \\
\end{align*}
\]
--- rule strictly_precedes_e

proof -

have \( \{ p \cdot \forall n. (\text{run_tick_count } p K_2 p) \leq (\text{run_tick_count_strictly } p K_1 p) \} \)

\( = \{ p. (\text{run_tick_count } p K_2 n) \leq (\text{run_tick_count_strictly } p K_1 n) \} \)

\& \( \{ p. \forall n. (\text{run_tick_count } p K_2 p) \leq (\text{run_tick_count_strictly } p K_1 p) \} \)

using forall_nat_setSuc[of \( \{ \lambda n. (\text{run_tick_count } p K_2 n) \leq (\text{run_tick_count_strictly } p K_1 n) \} \) ]

by simp

thus \( \text{thesis} \) by auto

qed

lemma TESL_interp_stepwise_kills_coind_unfold:

\[
\begin{align*}
\{ K_1 \text{ kills } K_2 \} & \text{TESL}^{\geq n} = \\
\bigl( \{ K_1 \not\subset n \} \bigr) & \text{prim} \\
\bigcup & \bigl( K_1 \text{ kills } K_2 \} & \text{TESL}^{\geq \text{Suc } n}\bigr) \\
\end{align*}
\]
--- rule kills_e1

proof -

let \( ?\text{kills} = (\lambda n. \forall p \geq n. \text{hamlet} ((\text{Rep_run } p) K_1)) \)

\( \rightarrow (\forall n \geq c. \text{hamlet} ((\text{Rep_run } p) n c)) \)

let \( ?\text{ticks} = (\lambda n. \forall c \cdot \forall n \geq \text{hamlet} ((\text{Rep_run } p) m c)) \)

have \( \{ K_1 \text{ kills } K_2 \} & \text{TESL}^{\geq n} = \{ p. ?\text{kills } p \} \)

by simp

also have \( \gamma = (\{ p . ?\text{ticks } p K_1 \} \cap \{ p . ?\text{kills } (\text{Suc } n) p \}) \)

\( \cup \{ p . ?\text{ticks } p K_1 \} \cap \{ p . ?\text{dead } n p K_2 \} \)

by blast

} thus \( \{ p. ?\text{kills } p \} \)

\( \subseteq \{ p. \not\exists ?\text{ticks } p K_1 \} \cap \{ p . ?\text{kills } (\text{Suc } n) p \}

\( \cup \{ p . ?\text{ticks } p K_1 \} \cap \{ p . ?\text{dead } n p K_2 \} \)

by blast

next

\{ fix \( p::('r::linordered_field) \)

assume \( p \in \{ p . ?\text{kills } n p \} \)

hence \( ?\text{kills } n p \) by simp

hence \( ([\text{ticks } n) \land \text{dead } n \land ?\text{kills } (\text{Suc } n) p) \)

using Suc_less by blast

hence \( p \in \{ q . ?\text{ticks } n \land \text{dead } n \land ?\text{kills } (\text{Suc } n) p \}

\( \cup \{ p . \text{?ticks } n \land ?\text{kills } (\text{Suc } n) p \} \)

by blast

} thus \( \{ p . ?\text{kills } n p \} \)

\( \subseteq \{ p. \not\exists ?\text{ticks } n K_1 \} \cap \{ q. ?\text{kills } (\text{Suc } n) q \}

\( \cup \{ q . ?\text{ticks } n K_1 \} \cap \{ q . ?\text{dead } n q K_2 \} \)

by blast

qed

also have \( \gamma = \{ p. \not\exists ?\text{ticks } n K_1 \} \cap \{ q. ?\text{kills } (\text{Suc } n) q \}

\( \cup \{ q . ?\text{ticks } n K_1 \} \cap \{ q . ?\text{dead } n q K_2 \} \)

using Collect_cong Collect_disj_eq by auto

also have \( \gamma = \{ K_1 \not\subset n \} \bigr) \cap \{ K_1 \text{ kills } K_2 \} & \text{TESL}^{\geq \text{Suc } n} \)
6.3. INTERPRETATION OF CONFIGURATIONS

The stepwise interpretation of a TESL formula is the intersection of the interpretation of its atomic components.

fun TESL_interpretation_stepwise ::
  'τ::linordered_field TESL_formula ⇒ nat ⇒ 'τ run set
where
  ⟨(λϕ. [ [ϕ] TESL≥n] )⟩
  by (induction ϕ, simp, auto)

The global interpretation of a TESL formula is its interpretation starting at the first instant.

fun HeronConf_interpretation ::
  'τ::linordered_field config ⇒ 'τ run set
where
  ⟨(λΓ1. n ⊢ Ψ1 ⊢ Φ1)⟩
  by (induction Γ1, simp, auto)

6.3 Interpretation of configurations

The interpretation of a configuration of the operational semantics abstract machine is the intersection of:

- the interpretation of its context (the past),
- the interpretation of its present from the current instant,
- the interpretation of its future from the next instant.
When there are no remaining constraints on the present, the interpretation of a configuration is the same as the configuration at the next instant of its future. This corresponds to the introduction rule of the operational semantics.

**Lemma HeronConf_interp_stepwise_instant_cases:**

\[ \Gamma, n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma, \text{Suc } n \vdash \Phi \]_{\text{config}}

**Proof**

- \[ \Gamma, n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}} by simp

- \[ \Gamma, n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}} by simp

- \[ \Gamma, n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}} by simp

The following lemmas use the unfolding properties of the stepwise denotational semantics to give rewriting rules for the interpretation of configurations that match the elimination rules of the operational semantics.

**Lemma HeronConf_interp_stepwise_sporadicon_cases:**

\[ \Gamma, \text{Suc } n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}}

**Proof**

- \[ \Gamma, \text{Suc } n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}} by simp

- \[ \Gamma, \text{Suc } n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}} by simp

- \[ \Gamma, \text{Suc } n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}} by simp

ultimately show \( \psi \) by blast

**QED**

The following lemmas use the unfolding properties of the stepwise denotational semantics to give rewriting rules for the interpretation of configurations that match the elimination rules of the operational semantics.

**Lemma HeronConf_interp_stepwise_tagrel_cases:**

\[ \Gamma, \text{Suc } n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}}

**Proof**

- \[ \Gamma, \text{Suc } n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}} by simp

- \[ \Gamma, \text{Suc } n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}} by simp

- \[ \Gamma, \text{Suc } n \vdash \Phi \]_{\text{config}} \Rightarrow \Gamma \vdash \Phi \]_{\text{config}} by simp

ultimately show \( \psi \) by blast

**QED**

(Proof details are not shown due to the complexity and length of the expressions.)
have \( \langle \Gamma, n \vdash (\text{time-relation } [K_1, K_2] \in R) \# \Psi \triangleright \Phi \rangle \rangle_{\text{config}} \)
- \( \langle \langle \Gamma \rangle_{\text{prim}} \cap \langle \text{time-relation } [K_1, K_2] \in R \rangle \# \Psi \rangle \parallel_{\text{TESL}} \geq n \) 
\( \cap \langle \Phi \rangle \parallel_{\text{TESL}} \geq \text{Succ } n \), by simp

moreover have \( \langle \langle \tau_{\text{var}}(K_1, n), \tau_{\text{var}}(K_2, n) \rangle \in R \# \Gamma \rangle, n \vdash \Psi \triangleright ((\text{time-relation } [K_1, K_2] \in R) \# \Phi \rangle \rangle_{\text{config}} \)
- \( \langle \langle \tau_{\text{var}}(K_1, n), \tau_{\text{var}}(K_2, n) \rangle \in R \# \Gamma \rangle \parallel_{\text{prim}} \cap \langle \text{time-relation } [K_1, K_2] \in R \rangle \# \Psi \rangle \parallel_{\text{TESL}} \geq n \) 
\( \cap \langle \Phi \rangle \parallel_{\text{TESL}} \geq \text{Succ } n \), by simp

ultimately show \(? \text{thesis}\) 

proof -
- have \( \langle \langle \tau_{\text{var}}(K_1, n), \tau_{\text{var}}(K_2, n) \rangle \in R \# \Gamma \rangle \parallel_{\text{prim}} \cap \langle \text{time-relation } [K_1, K_2] \in R \rangle \# \Psi \rangle \parallel_{\text{TESL}} \geq n \) 
using \text{TESL_interpreted_stepwise_tagrel_coinc_unfold} 
\text{TESL_interpretation_stepwise_cons_morph} by blast

thus \(? \text{thesis}\) by auto

qed

lemma \text{HeronConf_interp_stepwise_implies_cases}: 
\( \langle \langle \Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \triangleright \Phi \rangle \rangle_{\text{config}} \)
- \( \langle \langle \Gamma \rangle_{\text{prim}} \cap \langle (K_1 \text{ implies } K_2) \# \Psi \rangle \parallel_{\text{TESL}} \geq n \rangle \cap \langle \Phi \rangle \parallel_{\text{TESL}} \geq \text{Succ } n \), by simp

moreover have \( \langle \langle \Gamma \rangle_{\text{prim}} \cap \langle (K_1 \text{ implies } K_2) \# \Psi \rangle \parallel_{\text{TESL}} \geq n \rangle \parallel_{\text{TESL}} \geq \text{Succ } n \), by simp

moreover have \( \langle \langle \Gamma \rangle_{\text{prim}} \cap \langle (K_1 \text{ implies } K_2) \# \Psi \rangle \parallel_{\text{TESL}} \geq n \rangle \parallel_{\text{TESL}} \geq \text{Succ } n \), by simp

ultimately show \(? \text{thesis}\) 

proof -
- have \( \langle \langle \Gamma \rangle_{\text{prim}} \cap \langle (K_1 \text{ implies } K_2) \# \Psi \rangle \parallel_{\text{TESL}} \geq n \rangle \parallel_{\text{TESL}} \geq \text{Succ } n \), by simp

using \text{TESL_interpreted_stepwise_implies_coinc_unfold} 
\text{TESL_interpretation_stepwise_cons_morph} by blast

have \( \langle \langle \Gamma \rangle_{\text{prim}} \cap \langle (K_1 \text{ implies } K_2) \# \Psi \rangle \parallel_{\text{TESL}} \geq n \rangle \parallel_{\text{TESL}} \geq \text{Succ } n \), by simp

hence \( \langle \langle \Gamma \rangle_{\text{prim}} \cap \langle (K_1 \text{ implies } K_2) \# \Psi \rangle \parallel_{\text{TESL}} \geq n \rangle \parallel_{\text{TESL}} \geq \text{Succ } n \), using \text{f1} by (simp add: \text{Inf_left_commute inf_assoc})

thus \(? \text{thesis}\) by (simp add: \text{Int_Un_distrib2 inf_assoc})

qed

lemma \text{HeronConf_interp_stepwise_implies_not_cases}: 
\( \langle \langle \Gamma, n \vdash (K_1 \text{ implies not } K_2) \# \Psi \triangleright \Phi \rangle \rangle_{\text{config}} \)
- \( \langle \langle \Gamma \rangle_{\text{prim}} \cap \langle (K_1 \text{ implies not } K_2) \# \Psi \rangle \parallel_{\text{TESL}} \geq n \rangle \cap \langle \Phi \rangle \parallel_{\text{TESL}} \geq \text{Succ } n \), by simp

moreover have \( \langle \langle \Gamma \rangle_{\text{prim}} \cap \langle (K_1 \text{ implies not } K_2) \# \Psi \rangle \parallel_{\text{TESL}} \geq n \rangle \parallel_{\text{TESL}} \geq \text{Succ } n \), by simp
CHAPTER 6. SEMANTICS EQUIVALENCE

by simp
moreover have \[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \] _config
\[ \sim \]\[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \] _config
\[ \text{using } \text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} \]

ultimately show ?thesis

proof -

have \[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \] _config
\[ \sim \]\[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \] _config
\[ \text{using } \text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} \]

thus ?thesis by simp

qed

lemma HeronConf_interp_stepwise_timedelayed_cases:
\[ \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi \triangleright \Phi) \] _config
\[ \sim \]\[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \text{using } \text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} \]

ultimately show ?thesis

proof -

have \[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \sim \]\[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \text{using } \text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} \]

thus ?thesis by simp

qed

proof -

have \[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \sim \]\[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \text{using } \text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} \]

ultimately show ?thesis

proof -

have \[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \sim \]\[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \text{using } \text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} \]

ultimately show ?thesis

proof -

have \[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \sim \]\[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \text{using } \text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} \]

ultimately show ?thesis

proof -

have \[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \sim \]\[ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \] _config
\[ \text{using } \text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} \]

ultimately show ?thesis

proof -
6.3. INTERPRETATION OF CONFIGURATIONS

have \( \{ (K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \implies K_3) \# \Psi \}_\text{TESL} \succeq n \)
- \( \{ (K_1 \neg \uparrow n) \# \text{prim} \cup (K_1 \uparrow n) \# \text{prim} \cap (K_2 \#\# n + \delta \tau \Rightarrow K_1) \# \text{prim} \) 
\( \cap K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \implies K_3 \}_\text{TESL} \succeq n \)
using \text{TESL_interp_stepwise_timedelayed_coind_unfold} 
\text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} 
then show \( \text{thesis} \) 
by (simp add: Int_assoc Int_left_commut)
qed
then show \( \text{thesis} \) by (simp add: inf_assoc inf_sup_distrib2)
qed

lemma HeronConf_interp_stepwise_weakly_precedes_cases:
\begin{align*}
\Gamma, n \vdash (K_1 \text{ weakly precedes } K_2) \# \Psi \Increase \Phi \text{ config} \\
\Gamma \cap (K_1 \text{ weakly precedes } K_2) \# \Psi \}_\text{TESL} \succeq n \\
\cap \Phi \}_\text{TESL} \succeq n \) \text{ by simp} 
\end{align*}
moreover have \( \{ ((\# K_2 n, \# K_1 n) \in (\lambda(x,y). x \leq y)) \# \Gamma, n \)
- \( \Psi \Increase ((K_1 \text{ weakly precedes } K_2) \# \Phi) \text{ config} \)
\( \cap \Phi \}_\text{TESL} \succeq n \cap \Psi \}_\text{TESL} \succeq n \) 
by simp 
ultimately show \( \text{thesis} \)
- \( \text{proof} \)
- \( \text{have } \{ ((\# K_2 n, \# K_1 n) \in (\lambda(x,y). x \leq y)) \}_\text{prim} \)
\( \cap \text{K1 weakly precedes } K_2 \}_\text{TESL} \succeq n \cap \Psi \}_\text{TESL} \succeq n \)
- \( \text{using } \text{TESL_interp_stepwise_weakly_precedes_coind_unfold} 
\text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} \)
thus \( \text{thesis} \) by auto
qed

lemma HeronConf_interp_stepwise_strictly_precedes_cases:
\begin{align*}
\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi \Increase \Phi \text{ config} \\
\Gamma \cap (K_1 \text{ strictly precedes } K_2) \# \Psi \}_\text{TESL} \succeq n \\
\cap \Phi \}_\text{TESL} \succeq n \) \text{ by simp} 
\end{align*}
moreover have \( \{ ((\# K_2 n, \# K_1 n) \in (\lambda(x,y). x \leq y)) \# \Gamma, n \)
- \( \Psi \Increase ((K_1 \text{ strictly precedes } K_2) \# \Phi) \text{ config} \)
\( \cap \Phi \}_\text{TESL} \succeq n \cap \Psi \}_\text{TESL} \succeq n \) 
by simp 
ultimately show \( \text{thesis} \)
- \( \text{proof} \)
- \( \text{have } \{ ((\# K_2 n, \# K_1 n) \in (\lambda(x,y). x \leq y)) \}_\text{prim} \)
\( \cap \text{K1 strictly precedes } K_2 \}_\text{TESL} \succeq n \cap \Psi \}_\text{TESL} \succeq n \)
- \( \text{using } \text{TESL_interp_stepwise_strictly_precedes_coind_unfold} 
\text{TESL_interpretation_stepwise_cons_morph} \text{ by blast} \)
thus \( \text{thesis} \) by auto
qed
lemma HeronConf_interp_stepwise_kills_cases:

\[ \langle \langle \Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi \rangle \rangle_{\text{config}} = \langle \langle (K_1 \not\uparrow n) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rangle \rangle_{\text{config}} \]

\[ \cup \langle \langle (K_1 \uparrow n) \# (K_2 \not\uparrow n) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rangle \rangle_{\text{config}} \]

proof -

have \[ \langle \langle \Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi \rangle \rangle_{\text{config}} = \langle \langle \Gamma \rangle \rangle_{\text{prim}} \cap \langle \langle (K_1 \text{ kills } K_2) \# \Psi \rangle \rangle_{\text{TESL} \geq n} \cap \langle \langle \Phi \rangle \rangle_{\text{TESL} \geq \text{Suc } n} \]

by simp

moreover have \[ \langle \langle (K_1 \not\uparrow n) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rangle \rangle_{\text{config}} = \langle \langle (K_1 \not\uparrow n) \# \Gamma \rangle \rangle_{\text{prim}} \cap \langle \langle \Psi \rangle \rangle_{\text{TESL} \geq n} \]

by simp

moreover have \[ \langle \langle (K_1 \uparrow n) \# (K_2 \not\uparrow \geq n) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rangle \rangle_{\text{config}} = \langle \langle (K_1 \uparrow n) \# (K_2 \not\uparrow \geq n) \# \Gamma \rangle \rangle_{\text{prim}} \cap \langle \langle \Psi \rangle \rangle_{\text{TESL} \geq n} \]

by simp

ultimately show \( ? \)thesis

proof -

have \[ \langle \langle (K_1 \text{ kills } K_2) \# \Psi \rangle \rangle_{\text{TESL} \geq n} = \langle \langle (K_1 \not\uparrow n) \rangle \rangle_{\text{prim}} \uplus \langle \langle (K_1 \uparrow n) \rangle \rangle_{\text{prim}} \cap \langle \langle (K_2 \not\uparrow \geq n) \rangle \rangle_{\text{prim}} \]

\[ \cap \langle \langle (K_1 \text{ kills } K_2) \rangle \rangle_{\text{TESL} \geq \text{Suc } n} \cap \langle \langle \Psi \rangle \rangle_{\text{TESL} \geq n} \]

using \text{TESL}._\text{interp}\_\text{stepwise}\_\text{kills}\_\text{coind}\_\text{unfold} \text{TESL}._\text{interpretation}\_\text{stepwise}\_\text{cons}\_\text{morph} by blast

thus \( ? \)thesis by auto

qed

qed

end
Chapter 7

Main Theorems

theory Hygge_Theory
imports
  Corecursive_Prop
begin

Using the properties we have shown about the interpretation of configurations and the stepwise unfolding of the denotational semantics, we can now prove several important results about the construction of runs from a specification.

7.1 Initial configuration

The denotational semantics of a specification $\Psi$ is the interpretation at the first instant of a configuration which has $\Psi$ as its present. This means that we can start to build a run that satisfies a specification by starting from this configuration.

theorem solve_start:
  shows $\langle [\{ \Psi \}]_{TESL} = \{ \emptyset, 0 \vdash \emptyset \}, 0 \vdash \Psi \exists \emptyset \rangle$
proof
  have $\langle [\{ \Psi \}]_{TESL} = [\{ \Psi \}]_{TESL} \geq 0 \rangle$
    by (simp add: TESL_interpretation_stepwise_zero')
  moreover have $\langle [\{ \emptyset, 0 \vdash \emptyset \}]_{config} = [\{ \emptyset \}]_{prm} \cap [\{ \Psi \}]_{TESL} \geq 0 \cap [\{ \emptyset \}]_{TESL} \geq \text{Suc} 0 \rangle$
    by simp
  ultimately show $\text{thesis}$ by auto
qed

7.2 Soundness

The interpretation of a configuration $S_2$ that is a refinement of a configuration $S_1$ is contained in the interpretation of $S_1$. This means that by making successive choices in building the instants of a run, we preserve the soundness of the constructed run with regard to the original specification.

lemma sound_reduction:
  assumes $\langle (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \leftrightarrow (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) \rangle$
  shows $\langle [\{ \Gamma_1 \}]_{prm} \cap [\{ \Psi_1 \}]_{TESL} \geq n_1 \cap [\{ \Phi_1 \}]_{TESL} \geq \text{Suc} n_1$
    $\geq [\{ \Gamma_2 \}]_{prm} \cap [\{ \Psi_2 \}]_{TESL} \geq n_2 \cap [\{ \Phi_2 \}]_{TESL} \geq \text{Suc} n_2 \rangle$ (is $\text{P}$)
proof
  -
from assms consider
(a) \((\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \iff (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)\)
(b) \((\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \iff (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)\)
using operational_semantics_step.simps by blast
thus \(?thesis\)
proof (cases)
case a
thus \(?thesis\) by (simp add: operational_semantics_intro.simps)
next
case b thus \(?thesis\)
proof (rule operational_semantics_elim.cases)
fix \(\Gamma \vdash K_1 \triangleright K_2 \Psi \Phi\)
assume \(\langle(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma, n \vdash (K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi \triangleright \Phi)\)
and \(\langle(\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) = (\Gamma, n \vdash \Psi \triangleright ((K_2 \text{ sporadic } \tau \text{ on } K_2) \# \Phi)\rangle\)
thus \(?P\) using HeronConf_interpretation.simps by blast
next
fix \(\Gamma \vdash K_1 \triangleright K_2 \Psi \Phi\)
assume \(\langle(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma, n \vdash \text{time-relation } [K_1, K_2] \in R) \# \Psi \triangleright \Phi)\)
and \(\langle(\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((\tau \triangleright n) \# K_2 \triangleright n \# \Gamma, n \vdash \Psi \triangleright ((\text{time-relation } [K_1, K_2] \in R) \# \Phi)\rangle\)
thus \(?P\) using HeronConf_interpretation.simps by blast
next
fix \(\Gamma \vdash K_1 \triangleright K_2 \Psi \Phi\)
assume \(\langle(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \triangleright \Phi)\)
and \(\langle(\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((\text{implies not } n) \# K_2 \triangleright n \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi)\rangle\)
thus \(?P\) using HeronConf_interpretation.simps by blast
next
fix \(\Gamma \vdash K_1 \triangleright K_2 \Psi \Phi\)
assume \(\langle(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma, n \vdash (K_1 \text{ implies not } K_2) \# \Psi \triangleright \Phi)\)
and \(\langle(\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((\text{implies not not } n) \# K_2 \triangleright n \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi)\rangle\)
thus \(?P\) using HeronConf_interpretation.simps by blast
next
fix \(\Gamma \vdash K_1 \triangleright K_2 \Psi \Phi\)
assume \(\langle(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma, n \vdash (K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi \triangleright \Phi)\)
7.2. SOUNDNESS

\[ \langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = \]
\[ ((\langle K_1 \not\vdash n \rangle \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi)) \]

thus ?P using HeronConf_interp_stepwise_timedelayed_cases

HeronConf_interpretation.simps by blast

next

fix \( \Gamma \) \( n \) \( K_1 \)

assume \( \langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = \)
\[ ((\langle K_1 \not\vdash n \rangle \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi)) \]

thus ?P using HeronConf_interp_stepwise_timedelayed_cases

HeronConf_interpretation.simps by blast

next

fix \( \Gamma \) \( n \) \( K_1 \)

assume \( \langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = \)
\[ ((\langle K_2 \not\vdash \sigma \rangle \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi)) \]

thus ?P using HeronConf_interp_stepwise_strictly_precedes_cases

HeronConf_interpretation.simps by blast

next

fix \( \Gamma \) \( n \) \( K_1 \)

assume \( \langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = \)
\[ ((\langle K_1 \not\vdash n \rangle \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \]

thus ?P using HeronConf_interp_stepwise_kills_cases

HeronConf_interpretation.simps by blast

next

fix \( \Gamma \) \( n \) \( K_1 \)

assume \( \langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = \)
\[ ((\langle K_2 \not\vdash \sigma \rangle \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \]

thus ?P using HeronConf_interp_stepwise_kills_cases

HeronConf_interpretation.simps by blast

next

fix \( \Gamma \) \( n \) \( K_1 \)

assume \( \langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = \)
\[ ((\langle K_1 \not\vdash n \rangle \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \]

thus ?P using HeronConf_interp_stepwise_kills_cases

HeronConf_interpretation.simps by blast

qed

lemma sound_reduction':
assumes \( \langle S_1 \leftrightarrow S_2 \rangle \)
shows \( \langle \langle S_1 \rangle_{\text{config}} \supseteq \langle S_2 \rangle_{\text{config}} \rangle \)
proof -
have \( \forall s_1 s_2. (\langle s_2 \rangle_{\text{config}} \subseteq \langle s_1 \rangle_{\text{config}}) \lor \neg(\langle s_1 \leftrightarrow s_2 \rangle) \)
using sound_reduction by fastforce
thus ?thesis using assms by blast

qed

lemma sound_reduction_generalized:
assumes \( \langle S_1 \leftrightarrow^* S_2 \rangle \)
shows \( \langle \langle S_1 \rangle_{\text{config}} \supseteq \langle S_2 \rangle_{\text{config}} \rangle \)
proof -
We start by proving that any run that is denoted by a configuration $S$ at any number of steps.

7.3 Completeness

We will now show that any run that satisfies a specification can be derived from the initial configuration, at any number of steps.

We start by proving that any run that is denoted by a configuration $S$ is necessarily denoted by at least one of the configurations that can be reached from $S$.

```isar
theorem soundness:
proof
  assumes \( \langle I, 0 \vdash \Psi \triangleright I \rangle \mapsto^k S \)
  shows \( \langle \Psi \rangle \text{FESL} \geq \langle S \rangle_{\text{config}} \)
using assms sound_reduction_generalized solve_start by blast

From the initial configuration, a configuration $S$ obtained after any number $k$ of reduction steps denotes runs from the initial specification $\Psi$.

lemma complete_direct_successors:
proof (induct S)
case Nil
  show \( \langle \Gamma, n \vdash \Psi \triangleright \Phi \rangle_{\text{config}} \leq \bigcup_{X \in X_{\text{next}}} (\langle \Gamma, n \vdash \Psi \triangleright X \rangle_{\text{config}}) \)
using fastforce
next
case (Cons $\psi$ $\Psi$) thus \?case
proof (cases $\psi$
  case (SporadicOn $K_1 \tau K_2$) thus \?thesis
using HeronConf_interop_stepwise_sporadicon_cases
next
case (TagRelation $K_1 K_2 R$) thus \?thesis
using HeronConf_interop_stepwise_tagrel_cases
next
case (Implies $K_1 K_2$) thus \?thesis
```

complete_direct_successors':

shows \((S \in \text{config} \subseteq (\bigcup_{X \in C_{\text{next}}} S. [X \in \text{config}])\)

proof =
from HeronConf_interpretation_cases obtain \(\Gamma \vdash \Psi \Phi\)
where \(\text{config} \subseteq \Gamma \vdash \Psi \Phi\) by blast
with complete_direct_successors[of \(\Gamma \vdash \Psi \Phi\)] show \(?\text{thesis}\) by simp

qed

Therefore, if a run belongs to a configuration, it necessarily belongs to a configuration derived from it.

lemma branch_existence:

assumes \(\langle \exists S_1. (S_1 \vdash S_2) \land (\in \subseteq S_2) \rangle\)

proof =
from assms complete_direct_successors'[of \(\Gamma \vdash \Psi \Phi\)] have \(\in \subseteq \bigcup_{X \in C_{\text{next}}} S_1 \vdash [X \in \text{config}]\) by blast
hence \(\exists S_1 \in C_{\text{next}}. \in \subseteq S_1\) by simp
thus \(?\text{thesis}\) by blast

lemma branch_existence':

assumes \(\langle \exists S_1. (S_1 \vdash^* S_2) \land (\in \subseteq S_2) \rangle\)

proof (induct \(k\))
case 0
  thus \(?\text{thesis}\) by (simp add: assms)
CHAPTER 7. MAIN THEOREMS

next
  case (Suc k)
    thus ?case
      using branch_existence relpowp_Suc_I[of _] (operational_semantics_step)
    by blast
qed

Any run that belongs to the original specification $\Psi$ has a corresponding configuration $S$ at any number $k$ of reduction steps from the initial configuration. Therefore, any run that satisfies a specification can be derived from the initial configuration at any level of reduction.

theorem completeness:
assumes $’g$ $\in \{ [[ \Psi ]]_{FESL} \}$
shows $\exists S. ((\Gamma, 0 \vdash \Psi \triangleright \varnothing) \rightarrow^k S)$
  using assms branch_existence' solve_start blast

7.4 Progress

Reduction steps do not guarantee that the construction of a run progresses in the sequence of instants. We need to show that it is always possible to reach the next instant, and therefore any future instant, through a number of steps.

lemma instant_index_increase:
assumes $’g$ $\in \{ \Gamma, n \vdash \Psi \triangleright \Phi \}_\text{config}$
shows $\exists k. ((\Gamma, n \vdash \Psi \triangleright \Phi) \rightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
  using assms instant_i intro_part by fastforce
proof (insert assms, induct $\Psi$ arbitrary: $\Gamma \Phi$
  case (Nil $\Gamma \Phi$
    then show ?case
      using instant_i intro_part by fastforce
      moreover have $’g$ $\in \{ \Gamma, \text{Suc } n \vdash \Phi \}_\text{config}$
        by auto
      moreover have $’g$ $\in \{ \Gamma, \text{Suc } n \vdash \Phi \}_\text{config}$
        using assms Nil.prems calculation(2) by blast
      ultimately show ?thesis by blast
      qed
    next
    case (Cons $\psi \Psi$
      then show ?case
        using instant_i intro_part by fastforce
        have branches:
          $’g$ $\in \{ \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi \}_\text{config}$
          $\Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi$
          $\text{Suc } n \vdash \Psi_k \triangleright \Phi_k$
          $\Rightarrow \exists k. ((\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
          $’g$ $\in \{ \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \}_\text{config}$
          proof
            assume h1: $’g$ $\in \{ \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \}_\text{config}$
            hence $\exists k. ((\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
            $’g$ $\in \{ \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \}_\text{config}$
            qed
          qed
        qed
      qed
    qed
  qed

next

using h1 SporadicOn.prens by simp
from this obtain $\Gamma_k \Psi_k \Phi_k \ k$ where
\[ f_p:((\Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi)) \rightarrow^k (\Gamma', \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k) \] and $q \in [\Gamma_k, \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k]_{\text{config}}$ by blast
have \[ ((\Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi) \] by (simp add: elims_part sporadic_on_el)
\[ \rightarrow^k (\Gamma', \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k) \] using $\text{Suc } k (\Gamma_k, \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k)$ by auto
thus $\text{thesis}$ using $f_p$ by blast

qed

from branches SporadicOn.prens(2) have
\[ q \in [\Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi)]_{\text{config}} \] by simp
with $rc$ show $\text{thesis}$ by blast

next
case (TagRelation $K_1 \ K_2 \ R$)
have branches: \[ [\Gamma, n \vdash ((\text{time-relation } [K_1, K_2] \in R) \# \Psi) \triangleright \Phi]_{\text{config}} \] by auto
\[ \rightarrow^k (\Gamma', \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k) \] using HeronConf_intervspause_tagrel_cases by simp
thus $\text{case}$
proof
have $\exists \Gamma_k \Psi_k \Phi_k x.$
\[ (((\text{time-relation } [K_1, K_2] \in R) \# \Psi), n \vdash \Psi \triangleright ((\text{time-relation } [K_1, K_2] \in R) \# \Phi)) \rightarrow^k (\Gamma', \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k) \] using TagRelation.prens by simp
from this obtain $\Gamma_k \Psi_k \Phi_k x.$
where $f_p:(((\text{time-relation } [K_1, K_2] \in R) \# \Psi), n \vdash \Psi \triangleright ((\text{time-relation } [K_1, K_2] \in R) \# \Phi))$
CHAPTER 7. MAIN THEOREMS

\[ \omega^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]

and \( rc: \langle q \in [ \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k ]^{config} \rangle \) by blast

have \( pc: (\Gamma, n \vdash ((\text{time-relation } [K_1, K_2] \in \mathbb{R}) \# \Psi) \triangleright \Phi) \)

\[ \rightarrow ((\text{time-relation } [K_1, K_2] \in \mathbb{R}) \# \Psi) \]

by (simp add: elims_part tagrel_e)

hence \((\Gamma, n \vdash (\text{time-relation } [K_1, K_2] \in \mathbb{R}) \# \Psi) \triangleright \Phi) \)

\[ \rightarrow \text{Suc } k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]

using \( \text{fp relpow}_\text{Suc}_{\text{I2}} \) by auto

with \( rc \) show \(?thesis by blast\)

qed

case \((\text{Implies } K_1, K_2)\)

have branches: \[ (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi )^{config} \]

\[ = (\Gamma, n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi ))^{config} \]

\[ \cup (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi )^{config} \]

using \( \text{HeronConf_inter stepwise implies_cases by simp} \)

moreover have \( br_1: (q \in [ (\Gamma, \vdash ((K_1 \vdash n) \# \Gamma) \# \Psi) \triangleright (K_1 \text{ implies } K_2) \# \Phi )^{config} \]

\[ \rightarrow \exists \Gamma_k \Psi_k \Phi_k k \]

\[ (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi ) \]

\[ \rightarrow \omega^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]

using \( h_1 \) implies \( \text{prems by simp} \)

from this obtain \( \Gamma_k \Psi_k \Phi_k k \) where

\( \text{fp}: (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi )^{config} \)

\[ \rightarrow \omega^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]

and \( rc: (q \in [ \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k ]^{config} \rangle \) by blast

have \( pc: (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi )^{config} \)

\[ \rightarrow ((\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi )^{config} \]

by (simp add: elims_part implies_e1)

hence \((\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi ) \)

\[ \rightarrow \text{Suc } k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]

using \( \text{fp relpow}_\text{Suc}_{\text{I2}} \) by auto

with \( rc \) show \(?thesis by blast\)

qed

case \((\text{Implies } K_1, K_2)\)

moreover have \( br_2: (q \in [ (\Gamma, \vdash ((K_1 \vdash n) \# \Psi) \triangleright \Phi )^{config} \]

\[ \rightarrow \exists \Gamma_k \Psi_k \Phi_k k \]

\[ (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi )^{config} \]

\[ \rightarrow \omega^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]

using \( h_2 \) implies \( \text{prems by simp} \)

from this obtain \( \Gamma_k \Psi_k \Phi_k k \) where

\( \text{fp}: (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi )^{config} \)

\[ \rightarrow \omega^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]

and \( rc: (q \in [ \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k ]^{config} \rangle \) by blast

have \((\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi )^{config} \)

\[ \rightarrow ((\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi )^{config} \]

by (simp add: elims_part implies_e2)
7.4. PROGRESS

hence $((\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \supset \Phi) \iff k (\Gamma_k, \text{Suc } n \vdash \Psi_k \supset \Phi_k)$

using fp relpowp\_Suc\_I2 by auto

with rc show ?thesis by blast

qed

ultimately show ?case using Implies\_prems(2) by blast

next

case (Implies\_Not $K_1 K_2$)

have branches: $((\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \supset \Phi) \iff config$

= $((\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \supset \Phi) \iff config)$

$\cup ((\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \supset \Phi) \iff config)$

using HeronConf\_interp\_stepwise\_implies\_not\_cases by simp

moreover have br1: $\forall \xi \in \{ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Phi \rightarrow ((K_1 \text{ implies } K_2) \# \Psi) \supset \Phi) \iff config$ prove

$\forall \xi \in \{ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Phi \rightarrow ((K_1 \text{ implies } K_2) \# \Psi) \supset \Phi) \iff config$ prove

and br2: $\forall \xi \in \{ ((K_1 \rightarrow n) \# \Gamma), n \vdash \Phi \rightarrow ((K_1 \text{ implies } K_2) \# \Psi) \supset \Phi) \iff config$ prove

by simp

hence $((\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \supset \Phi) \iff Suc \ k (\Gamma_k, \text{Suc } n \vdash \Psi_k \supset \Phi_k)$

using fp relpowp\_Suc\_I2 by auto

with rc show ?thesis by blast

qed
ultimately show \( \text{case using \text{ImpliesNot.prems}(2)} \) by blast

next case (\text{TimeDelayedBy} \ K_1 \ \delta \tau \ K_2 K_3)

have branches:
\[
\begin{align*}
\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \Rightarrow \Phi \mid_{\text{config}} \\
= \exists \Gamma_k \Psi_k \Phi_k k.
\end{align*}
\]

\[
\begin{align*}
\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \mid_{\text{config}}
\end{align*}
\]

\[
\begin{align*}
\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \Rightarrow \Psi \mid_{\text{config}}
\end{align*}
\]

\[
\begin{align*}
\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \Rightarrow \Psi \Rightarrow \Psi_k \Rightarrow \Phi_k \mid_{\text{config}}
\end{align*}
\]

using \text{HeronConf_interp_stepwise_timedelayed_cases} by simp

moreover have br1:
\[
\begin{align*}
g \in \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \mid_{\text{config}}
\end{align*}
\]

proof -

assume h1: \( g \in \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \mid_{\text{config}} \)

then have \( \exists \Gamma_k \Psi_k \Phi_k k. \)
\[
\begin{align*}
\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \Rightarrow \Psi_k \Rightarrow \Phi_k \mid_{\text{config}}
\end{align*}
\]

using h1 \text{TimeDelayedBy.prems by simp}

from this obtain \( \Gamma_k \Psi_k \Phi_k k \)

where fp: \( ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \)
\[
\begin{align*}
\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \Rightarrow \Psi_k \Rightarrow \Phi_k \mid_{\text{config}}
\end{align*}
\]

and rc: \( g \in \Gamma_k, n \vdash \Psi_k \Rightarrow \Phi_k \mid_{\text{config}} \) by blast

have \( \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \Rightarrow \Phi \)
\[
\begin{align*}
 \Rightarrow \Psi \Rightarrow ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \Rightarrow \Psi_k \Rightarrow \Phi_k \mid_{\text{config}}
\end{align*}
\]

by (simp add: \text{elims_part_timedelayed_el})

hence \( \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \Rightarrow \Phi \)
\[
\begin{align*}
\Rightarrow \Psi \Rightarrow ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \Rightarrow \Psi_k \Rightarrow \Phi_k \mid_{\text{config}}
\end{align*}
\]

using fp relpow_Suc_I2 by auto

with rc show \( ? \)thesis by blast

qed

moreover have br2:
\[
\begin{align*}
g \in \Gamma, n \vdash ((K_1 \uparrow \downarrow n) \# (K_2 \# n \uplus \delta \tau \Rightarrow K_3) \# \Gamma) \Rightarrow \Psi_k \Rightarrow \Phi_k \mid_{\text{config}}
\end{align*}
\]

\[
\begin{align*}
\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \Rightarrow \Psi_k \Rightarrow \Phi_k \mid_{\text{config}}
\end{align*}
\]

\[
\begin{align*}
\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \Rightarrow \Psi_k \Rightarrow \Phi_k \Rightarrow \Phi_k \mid_{\text{config}}
\end{align*}
\]

proof -

assume h2: \( g \in \Gamma, n \vdash ((K_1 \uparrow \downarrow n) \# (K_2 \# n \uplus \delta \tau \Rightarrow K_3) \# \Gamma) \Rightarrow \Psi_k \Rightarrow \Phi_k \mid_{\text{config}} \)

then have \( \exists \Gamma_k \Psi_k \Phi_k k. \)
\[
\begin{align*}
((K_1 \uparrow \downarrow n) \# (K_2 \# n \uplus \delta \tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \Rightarrow ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \Rightarrow \Psi_k \Rightarrow \Phi_k \Rightarrow \Phi_k \mid_{\text{config}}
\end{align*}
\]

using h2 \text{TimeDelayedBy.prems by simp}

from this obtain \( \Gamma_k \Psi_k \Phi_k k \)

where fp: \( ((K_1 \uparrow \downarrow n) \# (K_2 \# n \uplus \delta \tau \Rightarrow K_3) \# \Gamma) \Rightarrow \Psi_k \Rightarrow \Phi_k \Rightarrow \Phi_k \mid_{\text{config}} \)

and rc: \( g \in \Gamma_k, n \vdash \Psi_k \Rightarrow \Phi_k \Rightarrow \Phi_k \mid_{\text{config}} \) by blast
have \((\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau) \text{ on } K_2 \text{ implies } K_3) \# \Psi \triangleright \Phi)\)
\[\iff ((K_1 \vdash n) \# (K_2 \circ n \oplus \delta \tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau) \text{ on } K_2 \text{ implies } K_3) \# \Phi)\]
by \((\text{simp add: elims_part timedelayed_e2})\)
\with \text{fp relpowp Suc_12 have}
\((\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau) \text{ on } K_2 \text{ implies } K_3) \# \Psi \triangleright \Phi)\)
\[\langle \text{Suc } k \quad (\Gamma_k, \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k)\rangle\]
by \text{auto}
\with \text{rc show ?thesis by blast}
\qd
ultimately show ?case using TimeDelayedBy.prems(2) by blast
\next
case \((\text{WeaklyPrecedes } K_1 K_2)\)
\have \[\langle \Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi \triangleright \Phi) \rangle_{\text{config}} = \]
\[\{ ((\# \leq K_2 n, \# \leq K_1 n) \in (\lambda(x, y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \} \langle \text{config} \rangle\]
using \text{HeronConf_interpol_stepwise_weakly_precedes_cases by simp}
moreover have \((q \in \Gamma, n \vdash ((\# \leq K_2 n, \# \leq K_1 n) \in (\lambda(x, y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \} \langle \text{config} \rangle\)
\[\Rightarrow (\exists \Gamma_k \Psi_k \Phi_k \text{ k. } ((\Gamma_k, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi))\]
\[\langle \text{Suc } k \quad (\Gamma_k, \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k)\rangle\]
using \text{WeaklyPrecedes.prems by simp}
from this obtain \(\Gamma_k \Psi_k \Phi_k \text{ k} \)
\where \text{fp:} (((\# \leq K_2 n, \# \leq K_1 n) \in (\lambda(x, y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi)\)
\[\langle \text{Suc } k \quad (\Gamma_k, \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k)\rangle\]
\and \text{rc:} \(q \in \Gamma, \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k \langle \text{config} \rangle \) by blast
\have \[(\Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi)\]
\[\langle \text{Suc } k \quad (\Gamma_k, \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k)\rangle\]
by \((\text{simp add: elims_part weakly_precedes_e})\)
\with \text{fp relpowp Suc_12 have}
\[(\Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi)\]
\[\langle \text{Suc } k \quad (\Gamma_k, \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k)\rangle\]
by \text{auto}
\with \text{rc show ?thesis by blast}
\qd
ultimately show ?case using \text{WeaklyPrecedes.prems(2)} by blast
\next
case \((\text{StrictlyPrecedes } K_1 K_2)\)
\have \[(\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi) \rangle_{\text{config}} = \]
\[\{ ((\# \leq K_2 n, \# < K_1 n) \in (\lambda(x, y). x < y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi) \} \langle \text{config} \rangle\]
using \text{HeronConf_interpolStepwise_strictly_precedes_cases by simp}
moreover have \((q \in \Gamma, n \vdash ((\# \leq K_2 n, \# < K_1 n) \in (\lambda(x, y). x < y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi) \} \langle \text{config} \rangle\)
\[\Rightarrow (\exists \Gamma_k \Psi_k \Phi_k \text{ k. } ((\Gamma_k, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi))\]
\[\langle \text{Suc } k \quad (\Gamma_k, \text{ Suc } n \vdash \Psi_k \triangleright \Phi_k)\rangle\]
\text{proof -}
\assume \((q \in \Gamma, n \vdash ((\# \leq K_2 n, \# < K_1 n) \in (\lambda(x, y). x < y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi) \} \langle \text{config} \rangle\)
hence $\exists \Gamma_k \Psi_k \Phi_k \; k$. $((\# \leq K_2 \; n, \# \leq K_1 \; n) \in \lambda(x, y), x \leq y)) \# \Gamma)$, n 
$\vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi))$ 
$\omega^k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)$)

using \text{StrictlyPrecedes}.prems by simp
from this obtain $\Gamma_k \Psi_k \Phi_k \; k$ where $fp:(((\# \leq K_2 \; n, \# \leq K_1 \; n) \in \lambda(x, y), x \leq y)) \# \Gamma)$, n 
$\vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi))$ 
$\omega^k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)$)

and $rc: \exists \rho \in \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \; \text{config}$ by blast
have $((\# = \#) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi))$
$\omega^k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)$

by (simp add: \text{elims_part_strictly_precedes_e})

with $fp$ \text{relpovp_Suc_I2} have $((\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi))$
$\omega^\text{Suc } k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$

by auto
with $rc$ show \text{thesis} by blast
qed

ultimately show \text{case using StrictlyPrecedes.prems}(2) by blast

next
case \text{(Kills } K_1 \text{ \_ \_ \_ } K_2)\text{ 

have branches: $\{ \Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi \} \text{config}$
$= \{ (\# \leq K_2 \; n, \# \leq K_1 \; n) \in \lambda(x, y), x \leq y) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi))$ 
$\omega^k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)$

using \text{HeronConf_interp_stepwise_kills_cases} by simp

moreover have br1: $\exists \rho \in \Gamma_k, \text{Suc } n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi)$
$\omega^k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)$

proof -

assume h1: $\exists \rho \in \Gamma_k, \text{Suc } n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi))$
then have $\exists \Gamma_k \Psi_k \Phi_k \; k$ where $fp:(((\# \leq K_2 \; n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi))$
$\omega^k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)$

and $rc: \exists \rho \in \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \; \text{config}$ by blast
have $pc:((\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi))$
$\omega^k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)$

by (simp add: \text{elims_part_kills_e1})

hence $((\# = \#) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi)$
$\omega^\text{Suc } k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$

using $fp$ \text{relpovp_Suc_I2} by auto
with $rc$ show \text{thesis} by blast
qed

moreover have br2:
$\exists \rho \in \Gamma_k, \text{Suc } n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi)$
$\omega^k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)$

proof -

assume h2: $\exists \rho \in \Gamma_k, \text{Suc } n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi))$
then have $\exists \Gamma_k \Psi_k \Phi_k \; k$. ( (\# = \#) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi))$
$\omega^k \; (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)$

) \land \rho \in \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \; \text{config}$
7.4. PROGRESS

using h2 Kills.prems by simp
drom this obtain Γk Ψk Φk k where

fp:(((K₁ ⊢ n) # (K₂ ⊢ n) # Γ), n ⊢ Ψ ⊢ ((K₁ kills K₂) # Φ))

⇒ (Γk, Suc n ⊢ Ψk ⊢ Φk)

and rc: q ∈ [[Γk, Suc n ⊢ Ψk ⊢ Φk] config] by blast

have ((Γ, n ⊢ ((K₁ kills K₂) # Ψ) ⊢ Φ), n ⊢ Ψ ⊢ ((K₁ kills K₂) # Φ))

⇒ (Γk, n ⊢ Ψk ⊢ Φk)

by (simp add: elim_part kills_e2)

hence ((Γ, n ⊢ ((K₁ kills K₂) # Ψ) ⊢ Φ) ⊢ Δ) ⋙ suc k (Γk, Suc n ⊢ Ψk ⊢ Φk)

using fp relpow_Suc_I2 by auto

with rc show "thesis" by blast

qed

ultimately show ?case using Kills.prems(2) by blast

qed

lemmas instant_index_increase_generalized:

assumes "n < n racially

assumes "q ∈ [[Γ, n ⊢ Ψ ⊢ Φ] config]

shows "∃Γk Ψk Φk k. ((Γ', n ⊢ Ψ ⊢ Φ) ⋙ k (Γk, n ⊢ Ψk ⊢ Φk))

∧ q ∈ [[Γk, δk + Suc n ⊢ Ψk ⊢ Φk] config]

proof -

obtain δk where diff: (n_k = δk + Suc n)

using add.commute assms(1) less_iff_Suc_add by auto

show "thesis"

proof (subdiff subst diff subst assms(2), insert assms(2), induct δ)

case 0 thus "case

using instant_index_increase assms(2) by simp

next

case (Suc δk)

have 0: "q ∈ [[Γ', n ⊢ Ψ ⊢ Φ] config] :=⇒ ∃Γk Ψk Φk k.

((Γ', n ⊢ Ψ ⊢ Φ) ⋙ k (Γk, n ⊢ Ψk ⊢ Φk))

∧ q ∈ [[Γk, n ⊢ Ψk ⊢ Φk] config]

using Suc.hyps by blast

obtain Γk Ψk Φk k where cont: (((Γ', n ⊢ Ψ ⊢ Φ) ⋙ k (Γk, δk + Suc n ⊢ Ψk ⊢ Φk))

∧ q ∈ [[Γk, δk + Suc n ⊢ Ψk ⊢ Φk] config]

using 0 cont instant_index_increase by blast

obtain Γ'k Ψ'k Φ'k k'

where cont2: (((Γk, δk + Suc n ⊢ Ψk ⊢ Φk)

⇒ k' (Γ'k, δk + Suc n ⊢ Ψk ⊢ Φk))

∧ q ∈ [[Γ'k, δk + Suc n ⊢ Ψk ⊢ Φk] config]

using Suc.prems using ccont contin by blast

have trans: ((Γ', n ⊢ Ψ ⊢ Φ) ⋙ k + k' (Γ'k, δk + Suc n ⊢ Ψk ⊢ Φk))

using operational_semantics_trans_generalized cont cont2 by blast

moreover have suc_assoc: (Suc δk + Suc n = Suc (δk + Suc n)) by arith

ultimately show "thesis"

proof (subdiff suc_assoc)

show "∃Γk Ψk Φk k.

((Γ', n ⊢ Ψ ⊢ Φ) ⋙ k (Γk, Suc (δk + Suc n) ⊢ Ψk ⊢ Φk))

∧ q ∈ [[Γk, Suc (δk + Suc n) ⊢ Ψk ⊢ Φk] config]

using cont2 local.trans by auto

qed

qed
Any run that belongs to a specification $\Psi$ has a corresponding configuration that develops it up to the $n^{th}$ instant.

**Theorem (progress):**

- Assumes $(\rho \in \{\Psi\})_{TESL}$
- Shows $\exists k \Gamma_k \Psi_k \Phi_k. ((\emptyset, 0 \vdash \Psi \triangleright \emptyset) \hookrightarrow^* (\Gamma_k, n \vdash \Psi_k \triangleright \Phi_k)) \land \rho \in \{ \Gamma_k, n \vdash \Psi_k \triangleright \Phi_k \}_{\text{config}}$

**Proof -**

- Have $1:\exists \Gamma_k \Psi_k \Phi_k. ((\emptyset, 0 \vdash \Psi \triangleright \emptyset) \hookrightarrow^* (\Gamma_k, 0 \vdash \Psi_k \triangleright \Phi_k)) \land \rho \in \{ \Gamma_k, 0 \vdash \Psi_k \triangleright \Phi_k \}_{\text{config}}$
- Using assms relpowp_0_I solve_start by fastforce

**Show ?thesis**

**Proof (cases $n = 0$)**

- Case True
  - Thus ?thesis using assms relpowp_0_I solve_start by fastforce
  - Next
    - Case False hence pos: $n > 0$ by simp
      - From assms solve_start have $(\rho \in \{\emptyset, 0 \vdash \Psi \triangleright \emptyset\}_{\text{config}})$ by blast
      - From instant_index_increase_generalized[OF pos this] show ?thesis by blast

**Qed**

### 7.5 Local termination

Here, we prove that the computation of an instant in a run always terminates. Since this computation terminates when the list of constraints for the present instant becomes empty, we introduce a measure for this formula.

**Primrec measure_interpretation :** $(\tau::\text{linordered_field TESL_formula} \Rightarrow \text{nat}) (\mu)$

where

- $\mu \emptyset = (0::\text{nat})$
- $\mu (\varphi \# \Phi) = (\text{case } \varphi \text{ of}$
  - sporadic _ on _ $\Rightarrow 1 + \mu \Phi$
  - _ $\Rightarrow 2 + \mu \Phi$)

**Fun measure_interpretation_config :** $(\tau::\text{linordered_field config} \Rightarrow \text{nat}) (\mu_{\text{config}})$

where

- $\mu_{\text{config}} (\Gamma, n \vdash \Psi \triangleright \Phi) = \mu (\Psi)$

We then show that the elimination rules make this measure decrease.

**Lemma elimination_rules_strictly_decreasing:**

- Assumes $(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow_e (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$
- Shows $\mu \Psi_1 > \mu \Psi_2$

**Using assms by (auto elim: operational_semantics_elim.cases)**

**Lemma elimination_rules_strictly_decreasing_means:**

- Assumes $(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow_e (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$
- Shows $(\Psi_2, \Psi_1) \in \text{measure } \mu$

**Using assms by (auto elim: operational_semantics_elim.cases)**

**Lemma elimination_rules_strictly_decreasing_means':**

- Assumes $\langle S_1 \rangle \hookrightarrow_e \langle S_2 \rangle$
- Shows $(\langle S_2, \mathcal{G}_2 \rangle) \in \text{measure } \mu_{\text{config}}$

**Proof -**

- From assms obtain $\Gamma_1 n_1 \Psi_1 \Phi_1$ where $p_1::\langle S_1 \rangle = (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1)$
  - Using measure_interpretation_config.cases by blast
- From assms obtain $\Gamma_2 n_2 \Psi_2 \Phi_2$ where $p_2::\langle S_2 \rangle = (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$

**Qed**
using measure_interpretation_config.cases by blast
from elimination_rules_strictly_decreasing_meas assms p1 p2
have \((\Psi_2, \Psi_1) \in \text{measure } \mu\) by blast
hence \(\mu \Psi_2 < \mu \Psi_1\) by simp
hence \(\mu_{\text{config}} (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) < \mu_{\text{config}} (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1)\) by simp
with p1 p2 show \(?thesis\) by simp
qed

Therefore, the relation made up of elimination rules is well-founded and the computation of an
instant terminates.

theorem instant_computation_termination:

\(\omega C (\lambda(S_1::'a::linordered_field config) S_2. (S_1 \hookrightarrow e \leftarrow S_2))\)

proof (simp add: \omegaC_def)
show \(\omega (((S_1::'a::linordered_field config), S_2). S_1 \hookrightarrow e \leftarrow S_2)\)
proof (rule \omegaC_subset)
  have \(\text{measure } \mu_{\text{config}} = ((S_2, (S_1::'a::linordered_field config)).\mu_{\text{config}} S_2 < \mu_{\text{config}} S_1)\)
    by (simp add: inv_image_def less_eq measure_def)
  thus \(\{((S_1::'a::linordered_field config), S_2). S_1 \hookrightarrow e \leftarrow S_2\} \subseteq (\text{measure } \mu_{\text{config}})\)
    using elimination_rules_strictly_decreasing_meas'
    operational_semantics_elim_inv_def by blast
next
show \(\omega (\text{measure measure_interpretation_config})\) by simp
qed
qed

end
Chapter 8

Properties of TESL

8.1 Stuttering Invariance

theory StutteringDefs

imports Denotational

begin

When composing systems into more complex systems, it may happen that one system has to perform some action while the rest of the complex system does nothing. In order to support the composition of TESL specifications, we want to be able to insert stuttering instants in a run without breaking the conformance of a run to its specification. This is what we call the *stuttering invariance* of TESL.

8.1.1 Definition of stuttering

We consider stuttering as the insertion of empty instants (instants at which no clock ticks) in a run. We characterize this insertion with a dilating function, which maps the instant indices of the original run to the corresponding instant indices of the dilated run. The properties of a dilating function are:

- it is strictly increasing because instants are inserted into the run,
- the image of an instant index is greater than it because stuttering instants can only delay the original instants of the run,
- no instant is inserted before the first one in order to have a well defined initial date on each clock,
- if \( n \) is not in the image of the function, no clock ticks at instant \( n \) and the date on the clocks do not change.

\[
\text{definition dilating_fun where}
\]

\[
\text{dilating_fun (f::nat ⇒ nat) (r::'a::linordered_field run)}
\equiv
\text{strict_mono f ∧ (f 0 = 0) ∧ (∀n. f n ≥ n ∧ (∀n₀. f n₀ = n) → (∀c. ¬(hamlet ((Rep_run r) n c))))}
\]

63
A run $r$ is a dilation of a run $\text{sub}$ by function $f$ if:

- $f$ is a dilating function for $r$
- the time in $r$ is the time in $\text{sub}$ dilated by $f$
- the hamlet in $r$ is the hamlet in $\text{sub}$ dilated by $f$

**definition dilating**

where

\[
\begin{align*}
\text{dilating } f \text{ sub } r & \equiv \text{dilating} \_ \text{fun } f \text{ r} \\
& \land (\forall c. \text{time} ((\text{Rep} \_ \text{run } \text{r}) (\text{Suc} \ n) c) \\
& = \text{time} ((\text{Rep} \_ \text{run } \text{r}) n c)) \\
& \land (\forall c. \text{hamlet} ((\text{Rep} \_ \text{run } \text{sub}) n c) = \text{hamlet} ((\text{Rep} \_ \text{run } \text{r}) (f n) c)))
\end{align*}
\]

A run is a subrun of another run if there exists a dilation between them.

**definition is_subrun**

where

\[
\begin{align*}
\text{is_subrun } :: (\forall a::\text{linordered_field} \ \text{run} \Rightarrow 'a \ \text{run} \Rightarrow \text{bool}) \ \text{infixl} \ ('\ll') \ 60
\end{align*}
\]

A contracting function is the reverse of a dilating function, it maps an instant index of a dilated run to the index of the last instant of a non stuttering run that precedes it. Since several successive stuttering instants are mapped to the same instant of the non stuttering run, such a function is monotonous, but not strictly. The image of the first instant of the dilated run is necessarily the first instant of the non stuttering run, and the image of an instant index is less that this index because we remove stuttering instants.

**definition contracting_fun**

where

\[
\begin{align*}
\text{contracting} \_ \text{fun } g \equiv \text{monotonic } g \land g \ 0 = 0 \land (\forall n. g \ n \leq n)
\end{align*}
\]

Figure 8.1 illustrates the relations between the instants of a run and the instants of a dilated run, with the mappings by the dilating function $f$ and the contracting function $g$.

A function $g$ is contracting with respect to the dilation of run $\text{sub}$ into run $r$ by the dilating function $f$ if:
8.1. STUTTERING INVARIANCE

- it is a contracting function;
- \((f \circ g) \ n\) is the index of the last original instant before instant \(n\) in run \(r\), therefore:
  - \((f \circ g) \ n \leq n\)
  - the time does not change on any clock between instants \((f \circ g) \ n\) and \(n\) of run \(r\);
  - no clock ticks before \(n\) strictly after \((f \circ g) \ n\) in run \(r\). See Figure 8.1 for a better understanding. Notice that in this example, \(2\) is equal to \((f \circ g) \ 2\), \((f \circ g) \ 3\), and \((f \circ g) \ 4\).

\[
\text{definition}
\text{contracting g r sub f} \equiv \text{contracting_fun g}
\land (\forall n. f (g \ n) \leq n)
\land (\forall c k. f (g \ n) \leq k \land k \leq n
\quad \rightarrow \text{time} ((\text{Rep_run} r) k c) = \text{time} ((\text{Rep_run sub}) (g \ n) c))
\land (\forall c k. f (g \ n) < k \land k \leq n
\quad \rightarrow \neg \text{hamlet} ((\text{Rep_run} r) k c))
\]

For any dilating function, we can build its inverse, as illustrated on Figure 8.1, which is a contracting function:

\[
\text{definition} \ (\text{dil_inverse} f :: (\text{n::nat} \Rightarrow \text{n}) \equiv (\lambda n. \text{Max} \{i. f i \leq n\}))
\]

8.1.2 Alternate definitions for counting ticks.

For proving the stuttering invariance of TESL specifications, we will need these alternate definitions for counting ticks, which are based on sets.

\[
\text{definition} \ (\text{tick_count} r c n :: (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \equiv (\lambda n. \text{card} \{i. i \leq n \land \text{hamlet} ((\text{Rep_run} r) i c)\})
\]

\[
\text{definition} \ (\text{tick_count_strict} r c n :: (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \equiv (\lambda n. \text{card} \{i. i < n \land \text{hamlet} ((\text{Rep_run} r) i c)\})
\]

8.1.3 Stuttering Lemmas

\[
\text{theory} \ \text{StutteringLemmas}
\]
\[
\text{imports} \ \text{StutteringDefs}
\]
\[
\text{begin}
\]

In this section, we prove several lemmas that will be used to show that TESL specifications are invariant by stuttering.
The following one will be useful in proving properties over a sequence of stuttering instants.

**Lemma bounded_suc_ind:**
- **Assumes:** \( \forall k. k < n \Rightarrow P (z + k) = P z \)
- **Shows:** \( k < n \Rightarrow P (z + k) = P z \)

**Proof** (induction on \( k \))
1. **Case 0**
   - With \( \text{assms}(1)[\text{of 0}] \) show \( \text{?case by simp} \)
2. **Case \( \text{Suc \ k}' \)**
   - With \( \text{assms}[\text{of \ Spec \ k}'] \) show \( \text{?case by force} \)

**8.1.4 Lemmas used to prove the invariance by stuttering**

Since a dilating function is strictly monotonous, it is injective.

**Lemma dilating_fun_injects:**
- **Assumes:** \( \text{dilating \_fun \ f \ r} \)
- **Shows:** \( \text{inj\_on \ f \ A} \)
- **Using:** \( \text{assms \ dilating\_fun\_def \ strict\_mono\_imp\_inj\_on \ by \ blast} \)

**Lemma dilating_injects:**
- **Assumes:** \( \text{dilating \ f \ sub \ r} \)
- **Shows:** \( \text{inj\_on \ f \ A} \)
- **Using:** \( \text{assms \ dilating\_def \ dilating\_fun\_injects \ by \ blast} \)

If a clock ticks at an instant in a dilated run, that instant is the image by the dilating function of an instant of the original run.

**Lemma ticks_image:**
- **Assumes:** \( \text{dilating \_fun \ f \ r} \) and \( \text{hamlet \ ((\text{Rep\_run} \ r) \ n \ c)} \)
- **Shows:** \( \exists n_0. f n_0 = n \)
- **Using:** \( \text{dilating\_fun\_def \ assms \ by \ blast} \)

**Lemma ticks_image_sub:**
- **Assumes:** \( \text{dilating \ f \ sub \ r} \) and \( \exists c. \text{hamlet \ ((\text{Rep\_run} \ r) \ n \ c)} \)
- **Shows:** \( \exists n_0. f n_0 = n \)
- **Using:** \( \text{dilating\_def \ ticks\_image \ by \ blast} \)

**Lemma ticks_image_sub':**
- **Assumes:** \( \text{dilating \ f \ sub \ r} \) and \( \exists c. \text{hamlet \ ((\text{Rep\_run} \ r) \ n \ c)} \)
- **Shows:** \( \exists n_0. f n_0 = n \)
- **Using:** \( \text{ticks\_image\_sub\[\text{OF \ assms}(1)] \ assms(2) \ by \ blast} \)

The image of the ticks in an interval by a dilating function is the interval bounded by the image of the bounds of the original interval. This is proven for all 4 kinds of intervals: \([m, n]\), \([m, n[\), \(]m, n]\) and \([m, n]\).

**Lemma dilating_fun_image_strict:**
- **Assumes:** \( \text{dilating \_fun \ f \ r} \)
- **Shows:** \( \langle k. f m < k \land k < f n \land \text{hamlet \ ((\text{Rep\_run} \ r) \ k \ c)} \rangle\)
- **Is:** \( \langle f (k. f m < k \land k < f n \land \text{hamlet \ ((\text{Rep\_run} \ r) \ (f k) \ c)}) \rangle \)
- **Proof**
  1. Fix \( k \) assume \( h: k \in \text{?IMG} \)
  2. From \( h \) obtain \( k_0 \) where \( \text{k0prop:(f \ k_0 = k \land \text{hamlet \ ((\text{Rep\_run} \ r) \ (f k_0) \ c)})} \)
8.1. STUTTERING INVARIANCE

using ticks_image[OF assms] by blast
with h have k ∈ image f ?SET:
  using assms dilating_fun_def strict_mono_less by blast
} thus (?IMG ⊆ image f ?SET) ..

next
{ fix k assume h: k ∈ image f ?SET:
  from h obtain k₀ where k₀prop: k = f k₀ ∧ k₀ ∈ ?SET) by blast
  hence (k ∈ ?IMG) using assms by (simp add: dilating_fun_def strict_mono_less)
} thus (image f ?SET ⊆ ?IMG) ..

qed

lemma dilating_fun_image_left:
assumes ⟨dilating_fun f r⟩
shows ⟨\{k. f m ≤ k ∧ k < f n ∧ hamlet ((Rep_run r) k c)\} = image f \{k. k ≤ m ∧ k ≤ n ∧ hamlet ((Rep_run r) (f k) c)\}⟩
(is ⟨?IMG = image f ?SET⟩)
proof
{ fix k assume h: k ∈ ?IMG:
  from h obtain k₀ where k₀prop: k = f k₀ ∧ k₀ ∈ ?SET) by blast
  using ticks_image[OF assms] by blast
  with h have k ∈ image f ?SET:
    using assms dilating_fun_def strict_mono_less strict_mono_less_eq by fastforce
} thus (?IMG ⊆ image f ?SET) ..

next
{ fix k assume h: k ∈ image f ?SET:
  from h obtain k₀ where k₀prop: k = f k₀ ∧ k₀ ∈ ?SET) by blast
  hence (k ∈ ?IMG) using assms dilating_fun_def strict_mono_less strict_mono_less_eq by fastforce
} thus (image f ?SET ⊆ ?IMG) ..

qed

lemma dilating_fun_image_right:
assumes ⟨dilating_fun f r⟩
shows ⟨\{k. f m < k ∧ k ≤ f n ∧ hamlet ((Rep_run r) k c)\} = image f \{k. m < k ∧ k ≤ n ∧ hamlet ((Rep_run r) (f k) c)\}⟩
(is ⟨?IMG = image f ?SET⟩)
proof
{ fix k assume h: k ∈ ?IMG:
  from h obtain k₀ where k₀prop: k = f k₀ ∧ k₀ ∈ ?SET) by blast
  using ticks_image[OF assms] by blast
  with h have k ∈ image f ?SET:
    using assms dilating_fun_def strict_mono_less strict_mono_less_eq by fastforce
} thus (?IMG ⊆ image f ?SET) ..

next
{ fix k assume h: k ∈ image f ?SET:
  from h obtain k₀ where k₀prop: k = f k₀ ∧ k₀ ∈ ?SET) by blast
  hence (k ∈ ?IMG) using assms dilating_fun_def strict_mono_less strict_mono_less_eq by fastforce
} thus (image f ?SET ⊆ ?IMG) ..

qed

lemma dilating_fun_image:
assumes ⟨dilating_fun f r⟩
shows ⟨\{k. f m ≤ k ∧ k ≤ f n ∧ hamlet ((Rep_run r) k c)\} = image f \{k. m ≤ k ∧ k ≤ n ∧ hamlet ((Rep_run r) (f k) c)\}⟩
(is ⟨?IMG = image f ?SET⟩)
proof
{ fix k assume h: k ∈ ?IMG:
  from h obtain k₀ where k₀prop: k = f k₀ ∧ k₀ ∈ ?SET) by blast
  using ticks_image[OF assms] by blast
  with h have k ∈ image f ?SET:
    using assms dilating_fun_def strict_mono_less strict_mono_less_eq by fastforce
} thus (?IMG ⊆ image f ?SET) ..

next
{ fix k assume h: k ∈ image f ?SET:
  from h obtain k₀ where k₀prop: k = f k₀ ∧ k₀ ∈ ?SET) by blast
  hence (k ∈ ?IMG) using assms dilating_fun_def strict_mono_less strict_mono_less_eq by fastforce
} thus (image f ?SET ⊆ ?IMG) ..

qed
On any clock, the number of ticks in an interval is preserved by a dilating function.

lemma ticks_as_often_strict:
assumes \( \text{dilating\_fun\ f\ r} \)
shows \( \langle \text{card} \{ p. n < p \land p < m \land \text{hamlet} ((\text{Rep\_run\ r}) (f p) c) \} = \text{card} \{ p. f n < p \land p < f m \land \text{hamlet} ((\text{Rep\_run\ r}) p c) \} \rangle \)
(is \( \langle \text{card} \ ?\text{SET} = \text{card} \ ?\text{IMG} \rangle \))
proof -
from dilating_fun_injects[OF assms] have \( \langle \text{inj\_on\ f\ ?\text{SET}} \rangle \) .
moreover have \( \langle \text{finite} \ ?\text{SET} \rangle \) by simp
from inj_on_iff_eq_card[OF this] calculation
have \( \langle \text{card} (\text{image} f \ ?\text{SET}) = \text{card} \ ?\text{SET} \rangle \) by blast
moreover from dilating_fun_image_strict[OF assms] have \( \langle ?\text{IMG} = \text{image} f \ ?\text{SET} \rangle \) .
ultimately show ?thesis by auto
qed

lemma ticks_as_often_left:
assumes \( \text{dilating\_fun\ f\ r} \)
shows \( \langle \text{card} \{ p. n \leq p \land p < m \land \text{hamlet} ((\text{Rep\_run\ r}) (f p) c) \} = \text{card} \{ p. f n < p \land p < f m \land \text{hamlet} ((\text{Rep\_run\ r}) p c) \} \rangle \)
(is \( \langle \text{card} \ ?\text{SET} = \text{card} \ ?\text{IMG} \rangle \))
proof -
from dilating_fun_injects[OF assms] have \( \langle \text{inj\_on\ f\ ?\text{SET}} \rangle \) .
moreover have \( \langle \text{finite} \ ?\text{SET} \rangle \) by simp
from inj_on_iff_eq_card[OF this] calculation
have \( \langle \text{card} (\text{image} f \ ?\text{SET}) = \text{card} \ ?\text{SET} \rangle \) by blast
moreover from dilating_fun_image_left[OF assms] have \( \langle ?\text{IMG} = \text{image} f \ ?\text{SET} \rangle \) .
ultimately show ?thesis by auto
qed

lemma ticks_as_often_right:
assumes \( \text{dilating\_fun\ f\ r} \)
shows \( \langle \text{card} \{ p. n < p \land p \leq m \land \text{hamlet} ((\text{Rep\_run\ r}) (f p) c) \} = \text{card} \{ p. f n < p \land p \leq f m \land \text{hamlet} ((\text{Rep\_run\ r}) p c) \} \rangle \)
(is \( \langle \text{card} \ ?\text{SET} = \text{card} \ ?\text{IMG} \rangle \))
proof -
from dilating_fun_injects[OF assms] have \( \langle \text{inj\_on\ f\ ?\text{SET}} \rangle \) .
moreover have \( \langle \text{finite} \ ?\text{SET} \rangle \) by simp
from inj_on_iff_eq_card[OF this] calculation
have \( \langle \text{card} (\text{image} f \ ?\text{SET}) = \text{card} \ ?\text{SET} \rangle \) by blast
moreover from dilating_fun_image_right[OF assms] have \( \langle ?\text{IMG} = \text{image} f \ ?\text{SET} \rangle \) .
ultimately show ?thesis by auto
qed

lemma ticks_as_often:
assumes \( \text{dilating\_fun\ f\ r} \)
shows \( \langle \text{card} \{ p. n \leq p \land p \leq m \land \text{hamlet} ((\text{Rep\_run\ r}) (f p) c) \} = \text{card} \{ p. f n \leq p \land p \leq f m \land \text{hamlet} ((\text{Rep\_run\ r}) p c) \} \rangle \)
8.1. STUTTERING INVARIANCE

(is (card ?SET = card ?IMG))

proof -
  from dilating_fun_inj[OOF asms] have (inj_on f ?SET).
  moreover have (finite ?SET) by simp
  from inj_on_iff_eq_card[OF this] calculation
  have (card (image f ?SET) = card ?SET) by blast
  moreover from dilating_fun_image[OF asms] have (?IMG = image f ?SET).
  ultimately show ?thesis by auto
qed

The date of an event is preserved by dilation.

lemma ticks_tag_image:
  assumes (dilating f sub r)
  and (∃c. hamlet ((Rep_run r) k c))
  and (time ((Rep_run r) k c) = τ)
  shows (∃k0. f k0 = k ∧ time ((Rep_run sub) k0 c) = τ)

proof -
  from ticks_image_sub'[OF asms(1,2)] have (∃k0. f k0 = k).
  from this obtain k0 where (f k0 = k) by blast
  moreover with asms(1,3) have (time ((Rep_run sub) k0 c) = τ)
    by (simp add: dilating_def)
  ultimately show ?thesis by blast
qed

TESL operators are invariant by dilation.

lemma ticks_sub:
  assumes (dilating f sub r)
  shows (hamlet ((Rep_run sub) n a) = hamlet ((Rep_run r) (f n) a))
  using asms
  by (simp add: dilating_def)

lemma no_tick_sub:
  assumes (dilating f sub r)
  shows (∀n0. f n0 = n) −→ ¬ hamlet ((Rep_run r) n a)
  using asms dilating_def dilating_fun_def
  by blast

Lifting a total function to a partial function on an option domain.

definition opt_lift :: ('a ⇒ 'a) ⇒ ('a option ⇒ 'a option)
  where 'opt_lift f ≡ λx. case x of None ⇒ None | Some y ⇒ Some (f y)

The set of instants when a clock ticks in a dilated run is the image by the dilation function of
the set of instants when it ticks in the subrun.

lemma tick_set_sub:
  assumes (dilating f sub r)
  shows {?S = image f ?R}

proof
  { fix k assume h:(k ∈ ?R)
    with no_tick_sub[OF asms] have (∃k0. f k0 = k) by blast
    from this obtain k0 where k0prop:(f k0 = k) by blast
    with ticks_sub[OF asms] have (hamlet ((Rep_run sub) k0 c)) by blast
    with k0prop have (k ∈ image f ?S) by blast
  }
  thus ?R ⊆ image f ?S) by blast
next
  { fix k assume h:(k ∈ image f ?S)
    from this obtain k0 where (f k0 = k ∧ hamlet ((Rep_run sub) k0 c)) by blast
  }
with assms have \( k \in ?R \) using \text{ticks}\_sub by blast
} thus \( \text{image} f ?S \subseteq ?R \) by blast qed

Strictly monotonous functions preserve the least element.

\textbf{lemma Least\_strict\_mono:}
\begin{align*}
\text{assumes } & \langle \text{strict\_mono } f \rangle \\
\text{and } & \langle \exists x \in S. \forall y \in S. x \le y \rangle \\
\text{shows } & \langle \text{LEAST } y. y \in f ' S = f \text{ (LEAST } x. x \in S) \rangle
\end{align*}
using \text{Least\_mono}[OF \text{strict\_mono}\_mono, OF assms].

A non-empty set of nats has a least element.

\textbf{lemma Least\_nat\_ex:}
\langle (n::nat) \in S = \Rightarrow \exists x \in S. (\forall y \in S. x \le y) \rangle
by (induction n rule: nat\_less\_induct, insert not\_le\_imp\_less, blast)

The first instant when a clock ticks in a dilated run is the image by the dilation function of the first instant when it ticks in the subrun.

\textbf{lemma Least\_sub:}
\begin{align*}
\text{assumes } & \langle \text{dilating } f \text{ sub } r \rangle \\
\text{and } & \langle \exists k\::\text{nat}. \text{hamlet } ((\text{Rep\_run sub}) k c) \rangle \\
\text{shows } & \langle \text{LEAST } k. k \in \{t. \text{hamlet } ((\text{Rep\_run r}) t c)\} = f \text{ (LEAST } k. k \in \{t. \text{hamlet } ((\text{Rep\_run sub}) t c)\}) \rangle
\end{align*}
proof -
from assms(2) have \( \exists x. x \in ?S \) by simp hence least: \( \exists x \in ?S. \forall y \in ?S. x \le y \)
using \text{Least\_nat\_ex}. from assms(1) have \( \text{strict\_mono } f \) by (simp add: \text{dilating}\_def \text{dilating}\_run\_def)
from \text{Least\_strict\_mono}[OF \text{this\_least}] have \( \langle \text{LEAST } y. y \in f ' ?S = f \text{ (LEAST } x. x \in ?S) \rangle \).
with \text{tick\_set}\_sub[OF assms(1), of \( c \)] show ?thesis by auto
qed

If a clock ticks in a run, it ticks in the subrun.

\textbf{lemma ticks\_imp\_ticks\_sub:}
\begin{align*}
\text{assumes } & \langle \text{dilating } f \text{ sub } r \rangle \\
\text{and } & \langle \exists k. \text{hamlet } ((\text{Rep\_run r}) k c) \rangle \\
\text{shows } & \langle \exists k_0. \text{hamlet } ((\text{Rep\_run sub}) k_0 c) \rangle
\end{align*}
proof -
from assms(2) obtain k where \( \text{hamlet } ((\text{Rep\_run r}) k c) \) by blast
with \text{ticks_image}\_sub[OF assms(1)] \text{ticks}\_sub[OF assms(1)] show ?thesis by blast
qed

Stronger version: it ticks in the subrun and we know when.

\textbf{lemma ticks\_imp\_ticks\_subk:}
\begin{align*}
\text{assumes } & \langle \text{dilating } f \text{ sub } r \rangle \\
\text{and } & \langle \text{hamlet } ((\text{Rep\_run r}) k c) \rangle \\
\text{shows } & \langle \exists k_0. f \ k_0 = k \land \text{hamlet } ((\text{Rep\_run sub}) k_0 c) \rangle
\end{align*}
proof -
from \text{no\_tick}\_sub[OF assms(1)] \text{assms}(2) have \( \exists k_0. f \ k_0 = k \) by blast
from this obtain \( k_0 \) where \( f \ k_0 = k \) by blast moreover with \text{ticks}\_sub[OF assms(1)] \text{assms}(2)
\text{have } \langle \text{hamlet } ((\text{Rep\_run sub}) k_0 c) \rangle \) by blast ultimately show ?thesis by blast
A dilating function preserves the tick count on an interval for any clock.

lemma dilated_ticks_strict:
assumes (dilating f sub r)
shows \((\forall i. f m < i < f n \land \text{hamlet } ((\text{Rep\_run } r) i c))\)
  = image f \((\{i. m < i < n \land \text{hamlet } ((\text{Rep\_run } \text{sub}) i c))\)\)
(is ?RUN = image f ?SUB)
proof
  { fix i assume h: (i \in \text{?SUB})
    hence \((m \leq i < n)\) by simp
    hence \((f m \leq f i < (f n))\) using assms
      by (simp add: dilating_def dilating_fun_def strict_monoD strict_mono_less_eq)
    moreover from h have \((\text{hamlet } ((\text{Rep\_run } r) f i c))\) using ticks_sub[OF assms] by blast
    ultimately have \((f i \in \text{?RUN})\) by simp
  }
  thus \((\text{image } f \text{ ?SUB} \subseteq \text{?RUN})\) by blast
next
  { fix i assume h: (i \in \text{?RUN})
    hence \((\text{hamlet } ((\text{Rep\_run } r) i c))\) by simp
    from ticks_imp_ticks_subk[OF assms this]
    obtain i0 where 10prop: \(f i0 = i \land \text{hamlet } ((\text{Rep\_run } \text{sub}) i0 c))\)
      by blast
    with h have \((f m \leq f i0 < f n)\) by simp
    moreover have \((\text{strict\_mono } f)\) using assms dilating_def dilating_fun_def by blast
    ultimately have \((m < i0 \land i0 < n)\)
      using strict_mono_less strict_mono_less_eq by blast
    with 10prop have \(\exists i0. f i0 = i \land i0 \in \text{?SUB}\) by blast
  }
  thus \((\text{?RUN} \subseteq \text{image } f \text{ ?SUB})\) by blast
qed

lemma dilated_ticks_left:
assumes (dilating f sub r)
shows \((\forall i. f m \leq i < f n \land \text{hamlet } ((\text{Rep\_run } r) i c))\)
  = image f \((\{i. m \leq i < n \land \text{hamlet } ((\text{Rep\_run } \text{sub}) i c))\)\)
(is ?RUN = image f ?SUB)
proof
  { fix i assume h: (i \in \text{?SUB})
    hence \((m \leq i \land i < n)\) by simp
    hence \((f m \leq f i \land f i < (f n))\) using assms
      by (simp add: dilating_def dilating_fun_def strict_monoD strict_mono_less_eq)
    moreover from h have \((\text{hamlet } ((\text{Rep\_run } r) f i c))\) using ticks_sub[OF assms] by blast
    ultimately have \((f i \in \text{?RUN})\) by simp
  }
  thus \((\text{image } f \text{ ?SUB} \subseteq \text{?RUN})\) by blast
next
  { fix i assume h: (i \in \text{?RUN})
    hence \((\text{hamlet } ((\text{Rep\_run } r) i c))\) by simp
    from ticks_imp_ticks_subk[OF assms this]
    obtain i0 where 10prop: \(f i0 = i \land \text{hamlet } ((\text{Rep\_run } \text{sub}) i0 c))\)
      by blast
    with h have \((f m \leq f i0 \land f i0 < f n)\) by simp
    moreover have \((\text{strict\_mono } f)\) using assms dilating_def dilating_fun_def by blast
    ultimately have \((m \leq i0 \land i0 < n)\)
      using strict_mono_less strict_mono_less_eq by blast
    with 10prop have \(\exists i0. f i0 = i \land i0 \in \text{?SUB}\) by blast
  }
  thus \((\text{?RUN} \subseteq \text{image } f \text{ ?SUB})\) by blast
qed

lemma dilated_ticks_right:
assumes \(\text{dilating f sub r}\)
shows \(\langle i. f n \leq i \land i \leq f n \land \text{hamlet} ((\text{Rep_run r}) i \ c)\rangle\)
\[\text{image f} \ (i. m \leq i \land i \leq n \land \text{hamlet} ((\text{Rep_run sub}) i \ c))\]
\(\text{(is \(?\text{RUN} = \text{image f} \ ?\text{SUB}\))}\)

proof
\{ fix \(i\) assume \(h: (i \in ?\text{SUB})\) \\
  hence \(m \leq i \land i \leq n\) by simp \\
  hence \(f m \leq f i \land f i \leq (f n)\) using assms by (simp add: dilating_def dilating_fun_def strict_monoD strict_mono_less_eq) \\
  moreover from \(h\) have \(\text{hamlet} ((\text{Rep_run sub}) i \ c)\) by simp \\
  hence \(\text{hamlet} ((\text{Rep_run r}) (f i) \ c)\) using \(\text{ticks_sub}[OF assms]\) by blast \\
  ultimately have \(f i \in ?\text{RUN}\) by simp \\
\} thus \(\text{image f} \ ?\text{SUB} \subseteq ?\text{RUN}\) by blast
next
\{ fix \(i\) assume \(h: (i \in ?\text{RUN})\) \\
  hence \(\text{hamlet} ((\text{Rep_run r}) i \ c)\) by simp \\
  from \(\text{ticks_imp_ticks_sub}[OF assms this]\) \\
  obtain \(i_0\) where \(i_0prop: f \ i_0 = i \land \text{hamlet} ((\text{Rep_run sub}) i_0 \ c)\) by blast \\
  with \(h\) have \(f m \leq f i_0 \land f i_0 \leq f n\) by simp \\
  using assms by (simp add: dilating_def dilating_fun_def strict_monoD strict_mono_less_eq) \\
  moreover from \(h\) have \(\text{hamlet} ((\text{Rep_run sub}) i_0 \ c)\) by simp \\
  hence \(\text{hamlet} ((\text{Rep_run r}) (f i_0) \ c)\) using \(\text{ticks_sub}[OF assms]\) by blast \\
  ultimately have \(f i_0 \in ?\text{RUN}\) by simp \\
\} thus \(?\text{RUN} \subseteq \text{image f} ?\text{SUB}\) by blast
qed

lemma dilated_ticks:
assumes \(\text{dilating f sub r}\)
shows \(\langle i. f n \leq i \land i \leq f n \land \text{hamlet} ((\text{Rep_run r}) i \ c)\rangle\)
\[\text{image f} \ (i. m \leq i \land i \leq n \land \text{hamlet} ((\text{Rep_run sub}) i \ c))\]
\(\text{(is \(?\text{RUN} = \text{image f} \ ?\text{SUB}\))}\)

proof
\{ fix \(i\) assume \(h: (i \in ?\text{SUB})\) \\
  hence \(m \leq i \land i \leq n\) by simp \\
  hence \(f m \leq f i \land f i \leq (f n)\) \\
  using assms by (simp add: dilating_def dilating_fun_def strict_monoD strict_mono_less_eq) \\
  moreover from \(h\) have \(\text{hamlet} ((\text{Rep_run sub}) i \ c)\) by simp \\
  hence \(\text{hamlet} ((\text{Rep_run r}) (f i) \ c)\) using \(\text{ticks_sub}[OF assms]\) by blast \\
  ultimately have \(f i \in ?\text{RUN}\) by simp \\
\} thus \(\text{image f} \ ?\text{SUB} \subseteq ?\text{RUN}\) by blast
next
\{ fix \(i\) assume \(h: (i \in ?\text{RUN})\) \\
  hence \(\text{hamlet} ((\text{Rep_run r}) i \ c)\) by simp \\
  from \(\text{ticks_imp_ticks_sub}[OF assms this]\) \\
  obtain \(i_0\) where \(i_0prop: f \ i_0 = i \land \text{hamlet} ((\text{Rep_run sub}) i_0 \ c)\) by blast \\
  with \(h\) have \(f m \leq f i_0 \land f i_0 \leq f n\) by simp \\
  using assms by (simp add: dilating_def dilating_fun_def strict_monoD strict_mono_less_eq) \\
  moreover from \(h\) have \(\text{hamlet} ((\text{Rep_run sub}) i_0 \ c)\) by simp \\
  hence \(\text{hamlet} ((\text{Rep_run r}) (f i_0) \ c)\) using \(\text{ticks_sub}[OF assms]\) by blast \\
  ultimately have \(f i_0 \in ?\text{RUN}\) by simp \\
\} thus \(?\text{RUN} \subseteq \text{image f} ?\text{SUB}\) by blast
qed

No tick can occur in a dilated run before the image of 0 by the dilation function.

lemma empty_dilated_prefix:
assumes \(\text{dilating f sub r}\)
and \(n < f 0\)
shows \(\neg \text{hamlet} ((\text{Rep_run r}) n \ c)\)

proof -
from assms have False by (simp add: dilating_def dilating_fun_def)
thus ?thesis ..
qed

corollary empty_dilated_prefix':
assumes "dilating f sub r"
shows "\{(i. f 0 \leq i \land i \leq f n \land \text{hamlet } ((\text{Rep}_r) i \ c)\} = \{(i. i \leq f n \land \text{hamlet } ((\text{Rep}_r) i \ c)\}"
proof -
from assms have "\{\text{strict_mono } f\}" by (simp add: dilating_def dilating_fun_def)
  hence \(\forall i. i \leq f n = (i < f 0) \lor (f 0 \leq i \land i \leq f n)\) by auto
  hence \(\{i. i \leq f n \land \text{hamlet } ((\text{Rep}_r) i \ c)\} = \{(i. i < f 0 \land i \leq f n \land \text{hamlet } ((\text{Rep}_r) i \ c))\}\) by auto
  also have \(\ldots = \{(i. f 0 \leq i \land i \leq f n \land \text{hamlet } ((\text{Rep}_r) i \ c)\}\) using empty_dilated_prefix'[OF assms]
  by blast
  finally show ?thesis by simp
qed

corollary dilated_prefix:
assumes "dilating f sub r"
shows "\{(i. i \leq f n \land \text{hamlet } ((\text{Rep}_r) i \ c)) = image f \{(i. 0 \leq i \land i \leq n \land \text{hamlet } ((\text{Rep}_r sub) i \ c))\}\}"
proof -
  have \(\{i. 0 \leq i \land i \leq f n \land \text{hamlet } ((\text{Rep}_r) i \ c)\} = image f \{(i. 0 \leq i \land i \leq n \land \text{hamlet } ((\text{Rep}_r sub) i \ c))\}\) using dilated_ticks[OF assms] empty_dilated_prefix'[OF assms]
  by blast
  thus ?thesis by simp
qed

corollary dilated_strict_prefix:
assumes "dilating f sub r"
shows "\{(i. i < f n \land \text{hamlet } ((\text{Rep}_r) i \ c)) = image f \{(i. i < n \land \text{hamlet } ((\text{Rep}_r sub) i \ c))\}\}"
proof -
  from assms have \(\{i. f 0 \leq i \land i \leq f n \land \text{hamlet } ((\text{Rep}_r) i \ c)\}\) unfolding dilating_def by simp
  from dilating_fun_image_left[OF dil, of \(\{0\} \ c\)] have \(\{i. f 0 \leq i \land i \leq n \land \text{hamlet } ((\text{Rep}_r) (f i) \ c)\}\) using dilating_fun_def by blast
  hence \(\{i. i < f n \land \text{hamlet } ((\text{Rep}_r) i \ c)\} = image f \{(i. i < n \land \text{hamlet } ((\text{Rep}_r sub) i \ c))\}\) using f0 by simp
  also have \(\ldots = image f \{(i. i < n \land \text{hamlet } ((\text{Rep}_r sub) i \ c))\}\) using assms dilating_def by blast
  finally show ?thesis by simp
qed

A singleton of \text{nat} can be defined with a weaker property.

lemma nat_sing_prop:
  \(\{(i::\text{nat}. i = k \land P(i)) = \{(i::\text{nat}. i = k \land P(k))\}\}\)
by auto

The set definition and the function definition of \text{tick_count} are equivalent.

lemma tick_count_is_fun[code]:\(\text{tick_count } r \ c \ n = \text{run_tick_count } r \ c \ n\)
proof (induction n)
CHAPTER 8. PROPERTIES OF TESL

74

case 0
have \( \langle \text{tick\_count } r c 0 = \text{card } i \cdot i \leq 0 \land \text{hamlet } ((\text{Rep\_run } r) i c) \rangle \)
  by (simp add: tick_count_def)
also have \( \langle \ldots = \text{card } i :: \text{nat}. i = 0 \land \text{hamlet } ((\text{Rep\_run } r) i c) \rangle \)
  using le_zero_eq nat_sing_prop[of 0 \( \lambda i. \text{hamlet } ((\text{Rep\_run } r) i c) \) by simp
also have \( \langle \ldots = \text{run\_tick\_count } r c 0 \rangle \) by simp
finally show ?case .
next
case (Suc k)
show ?case
proof (cases \( \langle \text{hamlet } ((\text{Rep\_run } r) (\text{Suc } k) c) \rangle \)
  case True
  hence \( \langle \{ i. i \leq \text{Suc } k \land \text{hamlet } ((\text{Rep\_run } r) i c) \} \rangle \)
    by auto
  hence \( \langle \text{tick\_count } r c (\text{Suc } k) = \text{Suc } (\text{tick\_count } r c k) \rangle \)
    by (simp add: tick_count_def)
  with Suc.IH have \( \langle \text{tick\_count } r c (\text{Suc } k) = \text{Suc } (\text{run\_tick\_count } r c k) \rangle \) by simp
  thus ?thesis by (simp add: True)
  next
  case False
  hence \( \langle \{ i. i \leq \text{Suc } k \land \text{hamlet } ((\text{Rep\_run } r) i c) \} \rangle \)
    by auto
  hence \( \langle \text{tick\_count } r c (\text{Suc } k) = \text{tick\_count } r c k \rangle \)
    by (simp add: tick_count_def)
  thus ?thesis using Suc.IH by (simp add: False)
next
qed

To show that the set definition and the function definition of \textit{tick\_count\_strict} are equivalent, we first show that the \textit{strictness} of \textit{tick\_count\_strict} can be softened using Suc.

**lemma** tick_count_strict_suc:**tick\_count\_strict r c (Suc n) = tick\_count r c n**
unfolding tick_count_strict_def tick_count_strict_def using less_Suc_eq_le by auto

**lemma** tick_count_strict_is_fun[code]:
\( \langle \text{tick\_count\_strict } r c n = \text{run\_tick\_count\_strictly } r c n \rangle \)
proof (cases \( \langle n = 0 \rangle \)
  case True
  hence \( \langle \text{tick\_count\_strict } r c n = 0 \rangle \) unfolding tick_count_strict_def by simp
  also have \( \langle \ldots = \text{run\_tick\_count\_strictly } r c 0 \rangle \)
    using run_tick_count_strictly.simps[of Suc m] by simp
  finally show ?thesis using True by simp
next
  case False
  from not0_implies_Suc[OF this] obtain m \( \langle s : m = \text{Suc } m \rangle \) by blast
  hence \( \langle \text{tick\_count\_strict } r c n = \text{tick\_count } r c m \rangle \)
    using tick_count_strictSuc by simp
  also have \( \langle \ldots = \text{run\_tick\_count\_strictly } r c m \rangle \)
    using run_tick_count_strictly.simps[of r c m] .
  also have \( \langle \ldots = \text{run\_tick\_count\_strictly } r c (\text{Suc } m) \rangle \)
    using run_tick_count_strictly.simps[of Suc m] .
  finally show ?thesis using * by simp
next
qed

This leads to an alternate definition of the strict precedence relation.

**lemma** strictly_precedes_alt_def1:
\( \langle (\varrho. \forall n :: \text{nat}. \text{run\_tick\_count } \varrho K_2 n) \leq \text{run\_tick\_count\_strictly } \varrho K_1 n \rangle \)
= \( \langle (\varrho. \forall n :: \text{nat}. \text{run\_tick\_count\_strictly } \varrho K_2 (\text{Suc } n) \rangle \)
The strict precedence relation can even be defined using only \texttt{run_tick_count}:

\begin{lstlisting}
lemma zero_gt_all:
proof
  Some properties of \texttt{run_tick_count}, \texttt{tick_count} and \texttt{Suc}: 
\end{lstlisting}
CHAPTER 8. PROPERTIES OF TESL

lemma run_tick_count_suc:
  ⟨run_tick_count r c (Suc n) = (if hamlet ((Rep_run r) (Suc n) c)
      then Suc (run_tick_count r c n)
      else run_tick_count r c n)⟩
by simp

corollary tick_count_suc:
  ⟨tick_count r c (Suc n) = (if hamlet ((Rep_run r) (Suc n) c)
      then Suc (tick_count r c n)
      else tick_count r c n)⟩
by (simp add: tick_count_is_fun)

Some generic properties on the cardinal of sets of nat that we will need later.

lemma card_suc:
  ⟨card {i::nat. i ≤ Suc n ∧ P i} = card {i. i ≤ n ∧ P i} + card {i. i = Suc n ∧ P i}⟩
proof -
  have {i::nat. i ≤ n ∧ P i} ∩ {i. i = Suc n ∧ P i} = {} by auto
  moreover have {i. i ≤ n ∧ P i} ∪ {i. i = Suc n ∧ P i} = {i. i ≤ Suc n ∧ P i}
    by auto
  moreover have :finite {i. i ≤ n ∧ P i} by simp
  moreover have :finite {i. i = Suc n ∧ P i} by simp
  ultimately show ?thesis using card_Un_disjoint[of {i. i ≤ n ∧ P i} {i. i = Suc n ∧ P i}] by simp
qed

lemma card_le_leq:
  assumes ⟨m < n⟩
  shows ⟨card {i::nat. i < n ∧ i ≤ n ∧ P i} = card {i. i < n ∧ i < n ∧ P i} + card {i. i = n ∧ P i}⟩
proof -
  have {i::nat. i < n ∧ i < n ∧ P i} ∩ {i. i = n ∧ P i} = {} by auto
  moreover with assms have {i::nat. i < n ∧ i < n ∧ P i} ∪ {i. i = n ∧ P i} = {i. m < i ∧ i ≤ n ∧ P i}
    by auto
  moreover have :finite {i. m < i ∧ i < n ∧ P i} by simp
  moreover have :finite {i. i = n ∧ P i} by simp
  ultimately show ?thesis using card_Un_disjoint[of {i. m < i ∧ i < n ∧ P i} {i. i = n ∧ P i}] by simp
qed

lemma card_le_leq_0:
  ⟨card {i::nat. i ≤ n ∧ P i} = card {i. i < n ∧ P i} + card {i. i = n ∧ P i}⟩
proof -
  have {i::nat. i ≤ n ∧ P i} ∩ {i. i = n ∧ P i} = {} by auto
  from assms have ∀i::nat. i < n = (i ≤ m) ∨ (m < i ∧ i < n)
    by auto
8.1. STUTTERING INVARIANCE

using less_trans by auto

hence 2:

\{i::nat. i < n \land P i\} = \{i. i \leq m \land P i\} \cup \{i. m \leq i \land i < n \land P i\}

by blast

have 3: finite \{i. i \leq m \land P i\} by simp

have 4: finite \{i. m \leq i \land i < n \land P i\} by simp

from card_Un_disjoint[OF 3 4 1] 2 show ?thesis by simp

qed

lemma card_mnm':
assumes \langle m < n \rangle
shows \langle card \{i::nat. i < n \land P i\} = card \{i. i < m \land P i\} + card \{i. m \leq i \land i < n \land P i\} \rangle

proof
-

have 1: \{i::nat. i < m \land P i\} \cap \{i. m \leq i \land i < n \land P i\} = {} by auto

from assms have \langle \forall i::nat. i < n = (i < m) \lor (m \leq i \land i < n) \rangle

using less_trans by auto

hence 2: \{i::nat. i < n \land P i\} = \{i. i < m \land P i\} \cup \{i. m \leq i \land i < n \land P i\}

by blast

have 3: finite \{i. i < m \land P i\} by simp

have 4: finite \{i. m \leq i \land i < n \land P i\} by simp

from card_Un_disjoint[OF 3 4 1] 2 show ?thesis by simp

qed

lemma nat_interval_union:
assumes \langle m \leq n \rangle
shows \langle \{i::nat. i \leq n \land P i\} = \{i. i \leq m \land P i\} \cup \{i::nat. m < i \land i \leq n \land P i\} \rangle

proof

using assms le_cases nat_less_le by auto

lemma card_sing_prop:
\langle card \{i. i = n \land P i\} = (if P n then 1 else 0) \rangle

proof

(cases \langle P n \rangle)

next

case False

hence \langle i. i = n \land P i\rangle = {} by (simp add: Collect_conv_if)

with \langle \neg P n \rangle show ?thesis by simp

qed

lemma card_prop_mono:
assumes \langle m \leq n \rangle
shows \langle card \{i::nat. i \leq m \land P i\} \leq card \{i. i \leq n \land P i\} \rangle

proof

from assms have \langle \{i. i \leq m \land P i\} \subseteq \{i. i \leq n \land P i\} \rangle by auto

moreover have finite \{i. i \leq n \land P i\} by simp

ultimately show ?thesis by (simp add: card_mono)

qed

In a dilated run, no tick occurs strictly between two successive instants that are the images by f of instants of the original run.

lemma no_tick_before_suc:
assumes \langle dilating f sub r \rangle

and \langle (f n) < k \land k < (f (Suc n)) \rangle

shows \langle \neg hamlet ((\text{Rep}_\text{run} r) k c) \rangle

proof

from assms(1) have smf: \langle strict_mono f \rangle by (simp add: dilating_def dilating_fun_def)

\{ fix k assume h: f n < k \land k < f (Suc n) \land hamlet ((\text{Rep}_\text{run} r) k c) \}


hence $\exists k_0. f \ k_0 = k$ using assms(1) dilating_def dilating_fun_def by blast
from this obtain $k_0$ where $f \ k_0 = k$ by blast
with h have $f \ n < f \ k_0 \land f \ k_0 < f \ (Suc \ n)$ by simp
hence False using smf not_less_eq strict_mono_less by blast
} thus ?thesis using assms(2) by blast
qed

From this, we show that the number of ticks on any clock at $f \ (Suc \ n)$ depends only on the number of ticks on this clock at $f \ n$ and whether this clock ticks at $f \ (Suc \ n)$. All the instants in between are stuttering instants.

lemma tick_count_fsuc:
assumes $dilating \ f \ sub \ r$
shows $\langle \text{tick\_count} \ r \ c \ (f \ (Suc \ n)) = \text{tick\_count} \ r \ c \ (f \ n) + \text{card} \ \{k. k = f \ (Suc \ n) \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\}\rangle$
proof -
have smf:$\langle \text{strict\_mono} \ f \rangle$ using assms dilating_def dilating_fun_def by blast
moreover have $\langle \text{finite} \ \{k. k \leq f \ n \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\}\rangle$ by simp
moreover have $\langle \text{finite} \ \{k. f \ n < k \land k \leq f \ (Suc \ n) \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\}\rangle$ by simp
ultimately have $\langle \text{strict\_mono\_less\_eq} \ \{k. k \leq f \ n \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\} = \{k. f \ n < k \land k \leq f \ (Suc \ n) \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\} \cup \{k. k = f \ (Suc \ n) \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\}\rangle$ by (simp add: nat_interval_union strict_mono_less_eq)
moreover have $\langle \text{card} \ \{k. f \ n < k \land k \leq f \ (Suc \ n) \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\} = 0\rangle$ by auto
ultimately have $\langle \text{card} \ \{k. k \leq f \ (Suc \ n) \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\} = \text{card} \ \{k. f \ n < k \land k \leq f \ (Suc \ n) \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\} \cup \{k. k = f \ (Suc \ n) \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\}\rangle$ by (simp add: * card_Un_disjoint)
moreover from no_tick_before_suc[OF assms] have $\langle \text{card} \ \{k. f \ n < k \land k \leq f \ (Suc \ n) \land \text{hamlet} \ ((Rep\_run \ r) \ k \ c)\} = 0\rangle$ by (simp add: * card_Un_disjoint)
ultimately show ?thesis by (simp add: tick_count_def)
qed

corollary tick_count_f_suc:
assumes $dilating \ f \ sub \ r$
shows $\langle \text{tick\_count} \ r \ c \ (f \ (Suc \ n)) = (\text{if} \ \text{hamlet} \ ((Rep\_run \ sub) \ (Suc \ n) \ c) \ \text{then} \ Suc \ (\text{tick\_count} \ r \ c \ (f \ n)) \ \text{else} \ \text{tick\_count} \ r \ c \ (f \ n))\rangle$
using tick_count_f_suc[DEF assms] by simp

corollary tick_count_f_suc_suc:
assumes $dilating \ f \ sub \ r$
shows $\langle \text{tick\_count} \ r \ c \ (f \ (Suc \ n)) = (\text{if} \ \text{hamlet} \ ((Rep\_run \ sub) \ (Suc \ n) \ c) \ \text{then} \ Suc \ (\text{tick\_count} \ r \ c \ (f \ n)) \ \text{else} \ \text{tick\_count} \ r \ c \ (f \ n))\rangle$
using tick_count_f_suc[DEF assms] by simp

lemma tick_count_f_suc_sub:
assumes $dilating \ f \ sub \ r$
shows $\langle \text{tick\_count} \ r \ c \ (f \ (Suc \ n)) = (\text{if} \ \text{hamlet} \ ((\text{Rep\_run} \ \text{sub}) (\Suc \ n) \ c) \ \text{then} \ Suc \ (\text{tick\_count} \ r \ c \ (f \ n)) \ \text{else} \ \text{tick\_count} \ r \ c \ (f \ n))\rangle$
using tick_count_f_suc[DEF assms] assms by (simp add: dilating_def)

The number of ticks does not progress during stuttering instants.
8.1. STUTTERING INVARIANCE

lemma tick_count_latest:
  assumes (dilating f sub r)
  and (f n p < n ∧ (∀k. f n k < k ∧ k ≤ n → (∃k0. f k0 = k))):
  shows (tick_count r c n = tick_count r c (f n p))
proof -
  have union: {i. i ≤ n ∧ hamlet ((Rep_run r) i c)} =
    {i. i ≤ f n p ∧ hamlet ((Rep_run r) i c)}
    ∪ {i. f n p < i ∧ i ≤ n ∧ hamlet ((Rep_run r) i c)}
using assms(2) by auto
  also have disjoint: {i. i ≤ f n p ∧ hamlet ((Rep_run r) i c)}
    ∩ {i. f n p < i ∧ i ≤ n ∧ hamlet ((Rep_run r) i c)} = {}
    by (simp add: disjoint_iff_not_equal)
from assms have {i. f n p < i ∧ i ≤ n ∧ hamlet ((Rep_run r) i c)} = {};
using no_tick_sub by fastforce
with union and partition show ?thesis by (simp add: tick_count_def)
qed

We finally show that the number of ticks on any clock is preserved by dilation.

lemma tick_count_sub:
  assumes (dilating f sub r)
  shows (tick_count sub c n = tick_count r c (f n))
proof -
  have tick_count sub c n = card {i. i ≤ n ∧ hamlet ((Rep_run sub) i c)}
    using tick_count_def[of ⟨sub⟩ ⟨c⟩ ⟨n⟩].
  also have ... = card (image f {i. i ≤ n ∧ hamlet ((Rep_run sub) i c)})
    using dilating_injects[OF assms] by (simp add: card_image)
  also have ... = card {i. i ≤ f n ∧ hamlet ((Rep_run r) i c)};
    using dilated_prefix[OF assms, symmetric, of ⟨⟨n⟩⟩ ⟨c⟩]
    by simp
  finally show ?thesis.
qed

corollary run_tick_count_sub:
  assumes (dilating f sub r)
  shows (run_tick_count sub c n = run_tick_count r c (f n))
proof -
  have run_tick_count sub c n = tick_count sub c n
    using tick_count_is_fun[of ⟨sub⟩ ⟨c⟩ ⟨n⟩, symmetric].
  also from tick_count_sub[OF assms] have (... = tick_count r c (f n)) .
  also have ... = #≤ r c (f n); using tick_count_is_fun[of r c ⟨f n⟩].
  finally show ?thesis .
qed

The number of ticks occurring strictly before the first instant is null.

lemma tick_count_strict_0:
  assumes (dilating f sub r)
  shows (tick_count_strict r c (f 0) = 0)
proof -
  from assms have (f 0 = 0) by (simp add: dilating_def dilating_fun_def)
  thus ?thesis unfolding tick_count_strict_def by simp
qed

The number of ticks strictly before an instant does not progress during stuttering instants.

lemma tick_count_strict_stable:
  assumes (dilating f sub r)
  assumes ((f n) < k ∧ k < (f (Suc n)))
  shows (tick_count_strict r c k = tick_count_strict r c (f (Suc n)))
proof -
from asms(1) have smf: ⟨strict_mono f⟩ by (simp add: dilating_def dilating_fun_def)
from asms(2) have ⟨f n < k⟩ by simp
hence ⟨∀i. k ≤ i → f n < i⟩ by simp
with no_tick_before_suc[OF asms(1)] have
⟨∀i. k ≤ i ∧ i < f (Suc n) → ¬ hamlet ((Rep_run r) i c)⟩ by blast
from tick_count_strict_def have
⟨tick_count_strict r c (f (Suc n)) = card {i. i < f (Suc n) ∧ hamlet ((Rep_run r) i c)} ⟩.
also have ⟨... = card {i. i < k ∧ hamlet ((Rep_run r) i c)}⟩ using * by simp
finally show ?thesis by (simp add: tick_count_strict_def)
qed

Finally, the number of ticks strictly before an instant is preserved by dilation.

lemma tick_count_strict_sub:
assumes ⟨dilating f sub r⟩
sows ⟨tick_count_strict sub c n = tick_count_strict r c (f n)⟩
proof -
have ⟨tick_count_strict sub c n = card {i. i < n ∧ hamlet ((Rep_run sub) i c)}⟩ using tick_count_def[of ⟨sub⟩ ⟨c⟩ ⟨n⟩] by simp
also have ⟨... = card (image f {i. i < n ∧ hamlet ((Rep_run sub) i c)})⟩ using asms dilating_def dilating_injects[OF asms] by (simp add: card_image)
also have ⟨... = card {i. i < f n ∧ hamlet ((Rep_run r) i c)}⟩ using dilated_strict_prefix[OF asms, symmetric, of ⟨r⟩ ⟨c⟩ ⟨n⟩] by simp
also have ⟨... = tick_count_strict r c (f n)⟩ using tick_count_def[of ⟨r⟩ ⟨c⟩ ⟨f n⟩] by simp
finally show ?thesis.
qed

The tick count on any clock can only increase.

lemma mono_tick_count:
⟨mono (λ k. tick_count r c k)⟩
proof
{ fix x y::nat
  assume ⟨x ≤ y⟩
  from card_prop_mono[OF this] have ⟨tick_count r c x ≤ tick_count r c y⟩ unfolding tick_count_def by simp
} thus ⟨∀x y. x ≤ y → tick_count r c x ≤ tick_count r c y⟩.
qed

In a dilated run, for any stuttering instant, there is an instant which is the image of an instant in the original run, and which is the latest one before the stuttering instant.

lemma greatest_prev_image:
assumes ⟨dilating f sub r⟩
sows ⟨(∃n₀. f n₀ = n) → (∃nₚ. f nₚ < n ∧ (∀k. f nₚ < k ∧ k ≤ n → (∃k₀. f k₀ = k)))⟩
proof (induction n)
case 0
  with asms have ⟨f 0 = 0⟩ by (simp add: dilating_def dilating_fun_def)
  thus ?case using "0.prems" by blast
next
case (Suc n)
show ?case
proof (cases ⟨∃n₀. f n₀ = n⟩)
  case True
from this obtain \( n_0 \) where \(( f \ n_0 = n)\) by blast

hence \(( f \ n_0 < (S\ n)) \land (\forall k. f \ n_0 < k \land k \leq (S\ n) \rightarrow (f k = k))\)

using \( S\ \text{prems} \ S\ \text{lne} \ \text{le\_antisym} \) by blast

thus \(?\text{thesis}\) by blast

next

case False

from \( S\ \text{IH}[OF this]\) obtain \( n_p \)

where \(( f \ n_p < n \land (\forall k. f \ n_p < k \land k \leq n \rightarrow (f k = k))\)\) by blast

hence \(( f \ n_p < S\ n \land (\forall k. f \ n_p < k \land k \leq n \rightarrow (f k = k))\)\) by simp

with \( S\ (2)\) have \(( f \ n_p < (S\ n) \land (\forall k. f \ n_p < k \land k \leq (S\ n) \rightarrow (f k = k)))\)

using \( \text{le\_Suc\_eq} \) by auto

thus \(?\text{thesis}\) by blast

qed

If a strictly monotonous function on \( \text{nat} \) increases only by one, its argument was increased only by one.

lemma \( \text{strict\_mono\_suc}\):

assumes \( \langle \text{strict\_mono}\ f \rangle\)

and \( \langle f \ n_p = Suc\ (f \ n)\rangle\)

shows \( \langle sn = Suc\ n\rangle\)

proof 

from \( \text{assm}(2)\) have \(( f \ n > f\ n)\) by simp

with \( \text{strict\_mono\_less}[OF \text{assm}(1)]\) have \(( f \ n > n)\) by simp

moreover have \(( f \ n \leq Suc\ n)\)

proof 

\{ assume \( f \ n > Suc\ n\)

from \( \text{assm}(1)\) obtain \( i\) where \( (n < i \land i < sn)\) by blast

hence \(( f \ n < f\ i \land f\ i < f\ sn)\) using \( \text{assm}(1)\) by \((\text{simp add: strict\_mono\_def})\)

with \( \text{assm}(2)\) have \( \text{False}\) by simp

\}

thus \(?\text{thesis}\) using \( \text{not\_less}\) by blast

qed

ultimately show \(?\text{thesis}\) by \((\text{simp add: Suc\_leI})\)

qed

Two successive non stuttering instants of a dilated run are the images of two successive instants of the original run.

lemma \( \text{next\_non\_stuttering}\):

assumes \( \langle \text{dilating}\ f\ \text{sub}\ r\rangle\)

and \( \langle f\ n_p < n \land (\forall k. f\ n_p < k \land k \leq n \rightarrow (f k = k))\rangle\)

and \( \langle f\ sn_0 = Suc\ n\rangle\)

shows \( \langle sn_0 \leq f\ (Suc\ n_p)\rangle\)

proof 

from \( \text{assm}(1)\) have \( \text{smf}/\langle\text{strict\_mono}\ f\rangle\) by \((\text{simp add: dilating\_def}\ \text{dilating\_fun\_def})\)

from \( \text{assm}(2)\) have \( \langle f\ n_p < n\rangle\) by simp

with \( \text{smf}\ \text{assm}(3)\) have \( \langle f\ sn_0 > n_p\rangle\) using \( \text{strict\_mono\_less}\) by fastforce

have \( \langle Suc\ n \leq f\ (Suc\ n_p)\rangle\)

proof 

\{ assume \( h:(Suc\ n > f\ (Suc\ n_p))\)

hence \( \langle Suc\ n_p < sn_0\rangle\) using \( \langle Suc\_less\rangle\ \text{assm}(3)\) by fastforce

hence \( \langle \exists k. k > n_p \land k < Suc\ n\rangle\) using \( h\) by blast

with \( \langle\rangle\) have \( \text{False}\) using \( \text{smf}\ \text{strict\_mono\_less}\) by blast

\}

thus \(?\text{thesis}\) using \( \text{not\_less}\) by blast

qed

hence \( \langle sn_0 \leq Suc\ n_p\rangle\) using \( \text{assm}(3)\) \text{smf}\ using \( \text{strict\_mono\_less\_eq}\) by fastforce

with \( \langle\rangle\) show \(?\text{thesis}\) by simp

qed
The order relation between tick counts on clocks is preserved by dilation.

\textbf{lemma dil\_tick\_count:}
assumes \((\text{sub} \ll \text{r})\) and \((\forall n. \text{run\_tick\_count sub a n} \leq \text{run\_tick\_count sub b n})\)
shows \((\text{run\_tick\_count r a n} \leq \text{run\_tick\_count r b n})\)
\[\text{proof -}
\text{from asms(1) is\_subrun\_def obtain f where } * : \text{dilating f sub r} \text{ by blast}
\text{show ?thesis}
\text{proof (induction n)}
\text{case 0}
\text{from asms(2) have } (\text{run\_tick\_count sub a 0} \leq \text{run\_tick\_count sub b 0}) \ldots
\text{with run\_tick\_count_sub[OF * , of _ 0] have}
(\text{run\_tick\_count r a (f 0)} \leq \text{run\_tick\_count r b (f 0)}) \text{ by simp}
\text{moreover from } * \text{ have } (f 0 = 0) \text{ by } (\text{simp add:dilating\_def dilating\_fun\_def})
\text{ultimately show ?case by simp}
\text{next}
\text{case } (\text{Suc n'}) \text{ thus ?case}
\text{proof (cases } (\exists n_0. \ f n_0 = \text{Suc n'}) \text{)}
\text{case True}
\text{from this obtain n_0 where fn0:(f n_0 = Suc n') by blast}
\text{show ?thesis}
\text{proof (cases } (\text{hamlet } ((\text{Rep\_run sub}) n_0 a)) \text{)}
\text{case True}
\text{have } (\text{run\_tick\_count r a (f n_0)} \leq \text{run\_tick\_count r b (f n_0)})
\text{using asms(2) run\_tick\_count_sub[OF *] by simp}
\text{thus ?thesis by } (\text{simp add: fn0})
\text{next}
\text{case False}
\text{hence } (\neg \text{hamlet } ((\text{Rep\_run r}) (\text{Suc n'}) a))
\text{using } * \text{ fn0 ticks\_sub by fastforce}
\text{thus ?thesis by } (\text{simp add: Suc.IH le\_SucI})
\text{qed}
\text{next}
\text{case False}
\text{thus ?thesis using } * \text{ Suc.IH no\_tick\_sub by fastforce}
\text{qed}
\text{qed}
\text{qed}

Time does not progress during stuttering instants.

\textbf{lemma stutter\_no\_time:}
assumes \((\text{dilating f sub r})\) and \((\forall k. \ f n < k \wedge k \leq n \implies (\exists k_0. \ f k_0 = k))\)
and \((m > f n)\)
shows \((\text{time } ((\text{Rep\_run r}) m c) = \text{time } ((\text{Rep\_run r}) (f n) c))\)
\[\text{proof -}
\text{from asms have } (\forall k. \ k < n = (f n) \implies (\exists k_0. \ f k_0 = \text{Suc } ((f n) + k))\text{ by simp}
\text{hence } (\forall k. \ k < n = (f n) \implies \text{time } ((\text{Rep\_run r}) (\text{Suc } ((f n) + k)) c) = \text{time } ((\text{Rep\_run r}) ((f n) + k) c))
\text{using asms(1) by } (\text{simp add: dilating\_def dilating\_fun\_def})
\text{hence } * : (\forall k. \ k < n = (f n) \implies \text{time } ((\text{Rep\_run r}) (\text{Suc } ((f n) + k)) c) = \text{time } ((\text{Rep\_run r}) (f n) c))
\text{using bounded\_succ\_ind[of } (m - (f n)) \langle\forall k. \ \text{time } ((\text{Rep\_run r}) (k c)) : (f n) \rangle \text{ by blast}
\text{from asms(3) obtain m_0 where m_0: } \text{Suc m_0} = m - (f n) \text{ using Suc\_diff\_Suc by blast}
\text{with } * \text{ have } \text{time } ((\text{Rep\_run r}) (\text{Suc } ((f n) + m_0)) c) = \text{time } ((\text{Rep\_run r}) (f n) c) \text{ by auto}
\text{moreover from m_0 have } \text{Suc } ((f n) + m_0) = m' \text{ by simp}
\text{ultimately show ?thesis by simp}
\text{qed}
8.1. STUTTERING INVARIANCE

lemma time_stuttering:
  assumes \(\text{dilating } f \text{ sub } r\)
  and \(\langle \text{time((Rep\_run sub n c) = } \tau \rangle\)
  and \(\forall k. f n < k \wedge k \leq m \implies (\exists k_0. f k_0 = k)\)
  and \(m > f n\)
  shows \(\langle \text{time((Rep\_run r m c) = } \tau \rangle\)
proof
  from assms(3) have \(\langle \text{time((Rep\_run r m c) = } \tau \rangle\)
  using stutter_no_time[OF assms(1,3,4)] by blast
  also from assms(1,2) have \(\langle \text{time((Rep\_run r (f n) c) = } \tau \rangle\)
  by (simp add: dilating_def)
  finally show \(?thesis\).
qed

The first instant at which a given date is reached on a clock is preserved by dilation.

lemma first_time_image:
  assumes \(\text{dilating } f \text{ sub } r\)
  shows \(\langle \text{first\_time sub c n t = first\_time r c (f n) t} \rangle\)
proof
  assume \(\langle \text{first\_time sub c n t} \rangle\)
  with before_first_time[OF this]
  have *: \(\langle \text{time((Rep\_run sub n c) = } t \wedge (\forall m < n. \text{time((Rep\_run sub m c) < } t))} \rangle\)
  by (simp add: first_time_def)
  moreover have \(\forall n c. \text{time(Rep\_run sub n c) = time(Rep\_run r (f n) c)}\)
  using assms(1) by (simp add: dilating_def)
  ultimately have **: \(\langle \text{time((Rep\_run r (f n) c) = } t \wedge (\forall m < n. \text{time((Rep\_run r m c) < } t))} \rangle\)
  by simp
  have \(\forall m < f n. \text{time((Rep\_run r m c) < } t)\)
  proof
    -
    { fix m assume hyp: \(m < f n\)
      have \(\langle \text{time((Rep\_run r m c) < } t) \rangle\)
      proof (cases \(\exists m_0. f m_0 = m\))
        case True
        from this obtain m_0 where mn0: \(m = f m_0\)
        with hyp have m0n: \(m_0 < n\)
        using assms(1)
        by (simp add: dilating_def dilating_fun_def strict_mono_less)
        hence \(\langle \text{time((Rep\_run sub m_0 c) < } t) \rangle\)
        using * by blast
        thus \(?thesis\) by (simp add: mn0 m0n **)
      next
        case False
        hence \(\exists m_p. f m_p < m \wedge (\forall k. f m_p < k \wedge k < m \implies (\exists k_0. f k_0 = k))\)
        using greatest_prev_image[OF assms] by simp
        from this obtain m_p where
        mp: \(f m_p < m \wedge (\forall k. f m_p < k \wedge k < m \implies (\exists k_0. f k_0 = k))\)
        by blast
        hence \(\langle \text{time((Rep\_run r m c) = time((Rep\_run sub m_p c))} \rangle\)
        using time_stuttering[OF assms] by blast
        also from hyp mp have \(f m_p < f n\)
        by linarith
        hence \(\langle \forall m_p < f n. \text{time((Rep\_run sub m_p c) < } t) \rangle\)
        by simp
        finally show \(?thesis\)
      qed
    } thus \(?thesis\) by simp
  qed
with ** show \(\langle \text{first\_time r c (f n) t} \rangle\)
  by (simp add: alt_first_time_def)
next
  assume \(\langle \text{first\_time r c (f n) t} \rangle\)
  hence \(\langle \forall k < f n. \text{time((Rep\_run r k c) < } t) \rangle\)
CHAPTER 8. PROPERTIES OF TESL

The first instant of a dilated run is necessarily the image of the first instant of the original run.

**Lemma first_dilated_instant:**

Assuming

- \( \text{dilating } f \) sub \( r \)
- \( n_0 = \text{Max} \{ i. f \cdot i \leq n \} \)
- \( f \cdot n_0 < n \) for all \( n > n_0 \)
- \( f \cdot n_0 > 0 \) for all \( n > n_0 \)

shows

- \( \{ i. f \cdot i \leq n \} = \{ n \} \)

**Proof:**

From assumptions (2) have \( \forall n > 0. f \cdot n > 0 \) by \( \text{simp add: finite_less_ub fge} \)

Hence \( \forall n \neq 0. \neg (f \cdot n \leq 0) \) by \( \text{simp add: dilating_def dilating_fun_def} \)

With assumptions (2) show \( (i. f \cdot i \leq 0) = \{ 0 \} \) by \( \text{blast} \)

Thus \( \text{thesis by simp} \)

**QED**

For any instant \( n \) of a dilated run, let \( n_0 \) be the last instant before \( n \) that is the image of an original instant. All instants strictly after \( n_0 \) and before \( n \) are stuttering instants.

**Lemma not_image_stut:**

Assuming

- \( \text{dilating } f \) sub \( r \)
- \( n_0 = \text{Max} \{ i. f \cdot i \leq n \} \)
- \( f \cdot n_0 < k \wedge k \leq n \)

shows

- \( \nexists k_0. f \cdot k_0 = k \)

**Proof:**

From assumptions (1) have

- \( \text{strict_mono } f \)
- \( \text{fxge}: \forall x. f \cdot x \geq x \)

Hence \( \forall n \neq 0. \neg (f \cdot n \leq 0) \) by \( \text{simp add: dilating_def dilating_fun_def} \)

By \( \text{auto simp: not_le} \)

With assumptions (3) \( \text{strict_mono_less[OF smf]} \) show \( \text{thesis by auto} \)

**QED**

For any dilating function \( f \), \( \text{dil_inverse } f \) is a contracting function.

**Lemma contracting_inverse:**

Assuming \( \text{dilating } f \) sub \( r \)

Shows \( \text{contracting } (\text{dil_inverse } f) \) r sub \( f \)

**Proof:**

From assumptions have

- \( \text{mono } (\text{dil_inverse } f) \)
- \( \text{no_img_tick}: \forall x. (f \cdot k_0 = k) \rightarrow (\forall c. \neg \text{hamlet } ((\text{Rep_run } r) k c)) \)
- \( \text{no_img_time}: \forall n. (f \cdot n_0 = (\text{Suc } n)) \rightarrow (\forall c. \text{time } ((\text{Rep_run } r) (\text{Suc } n) c) = \text{time } ((\text{Rep_run } r) n c)) \)

By \( \text{auto simp add: dilating_def dilating_fun_def} \)

Have \( \text{finite_prefix}: \forall n. \text{finite } (i. f \cdot i \leq n) \) by \( \text{auto simp add: finite_less_ub fge} \)

Have \( \text{prefix_not_empty}: \forall n. (i. f \cdot i \leq n) \neq \{ \} \) using \( \text{fom} \) by \( \text{blast} \)

Fix \( x :: \text{nat} \) and \( y :: \text{nat} \) assume \( \text{hyp: } (x \leq y) \)

Hence \( \text{inc: } (i. f \cdot i \leq x) \subseteq (i. f \cdot i \leq y) \)

**QED**
by (simp add: hyp Collect_mono le_trans)
from Max_mono[OF inc prefix_not_empty finite_prefix]
  have \((\text{dil\_inverse } f) \ x \leq (\text{dil\_inverse } f) \ y\) unfolding dil_inverse_def.
} thus \(\text{thesis}\) unfolding mono_def by simp
qed

from first_dilated_instant[OF smf f0] have 2:\((\text{dil\_inverse } f) \ 0 = 0\)
  unfolding dil_inverse_def.
from fxge have \(\forall \ n \ i. \ f \ i \leq \ n \rightarrow i \leq n\) using le_trans by blast
hence 3:\(\forall n. (\text{dil\_inverse } f) \ n \leq n\) using Max_in[OF finite_prefix prefix_not_empty]
  unfolding dil_inverse_def by blast
from 1 2 3 have *:\((\text{contracting\_fun } (\text{dil\_inverse } f))\) by (simp add: contracting_fun_def)
have \(\forall n. \text{finite } \{i. f i \leq n\}\) by (simp add: finite_prefix)
moreover have \(\forall n. (\text{finite } \{i. f i \leq n\}) \neq (\emptyset)\) using prefix_not_empty by blast
ultimately have 4:\(\forall n. (\text{dil\_inverse } f) \ n \leq n\)
  unfolding dil_inverse_def
  using asms(1) dilating_def dilating_fun_def Max_in by blast
have 5:\(\forall c k. f ((\text{dil\_inverse } f) \ n) < k 
  \land k \leq n\)
  \(\rightarrow \neg \text{hamlet } ((\text{Rep\_run } r) \ k \ c)\)
  using not_image_stut[OF asms] no_img_tick unfolding dil_inverse_def by blast
have 6:\(\forall c k. (\text{dil\_inverse } f) \ n \leq k 
  \land k \leq n\)
  \(\rightarrow \text{time } ((\text{Rep\_run } r) \ k \ c) = \text{time } ((\text{Rep\_run } sub) ((\text{dil\_inverse } f) \ n) \ c))\)
proof -
  { fix n c k assume h:\((\text{dil\_inverse } f) \ n \leq k \land k \leq n\)
    let \(\tau\) = \(\text{time } ((\text{Rep\_run } sub) ((\text{dil\_inverse } f) \ n) \ c))\)
    have tau:
      \(\text{time } ((\text{Rep\_run } sub) ((\text{dil\_inverse } f) \ n) \ c) = \tau)\) ..
    have gn:\((\text{dil\_inverse } f) \ n = \text{Max } \{i. f i \leq n\}\)
      unfolding dil_inverse_def ..
    from time_stuttering[OF asms tau, of k] not_image_stut[OF asms gn]
    have \(\text{time } ((\text{Rep\_run } r) \ k \ c) = \text{time } ((\text{Rep\_run } sub) ((\text{dil\_inverse } f) \ n) \ c))\)
      proof (cases \(f ((\text{dil\_inverse } f) \ n) = k\))
        case True
        moreover have \(\forall c k. \text{time } ((\text{Rep\_run } sub \ n \ c) = \text{time } ((\text{Rep\_run } r \ (f n) \ c))\)
        using asms by (simp add: dilating_def)
        ultimately show \text{thesis}\) by simp
next
  case False
  with h have \(f (\text{Max } \{i. f i \leq n\}) < k \land k \leq n\) by (simp add: dil_inverse_def)
  with time_stuttering[OF asms tau, of k] not_image_stut[OF asms gn]
  show \text{thesis}\) unfolding dil_inverse_def by auto
qed
} thus \text{thesis}\) by simp
qed

from * 4 5 6 show \text{thesis}\ unfolding contracting_def by simp
qed

The only possible contracting function toward a dense run (a run with no empty instants) is the
inverse of the dilating function as defined by \text{dil\_inverse}.

lemma dense_run_dil_inverse_only:
  assumes \((\text{dilating } f \ sub \ r)\) \(\land (\text{contracting } g \ r \ sub \ f)\)
  \(\land (\text{dense\_run } sub)\)
  shows \((g = (\text{dil\_inverse } f))\)
proof
  from assms(1) have \( \ast : \forall n. \text{finite} \{ i. f i \leq n \} \)
  using finite_less_ub by (simp add: dilating_def dilating_fun_def)
  from assms(1) have \( \forall 0 = 0 \) by (simp add: dilating_def dilating_fun_def)
  hence \( \forall n. 0 \in \{ i. f i \leq n \} \) by simp
  hence \( \ast : \forall n. (i. f i \leq n) \neq \emptyset \) by blast
  \{ fix \( n \) assume \( \exists k > g n \) unfolding dil_inverse_def using Max_in[OF * **]
  by blast \}
  from this obtain \( k \) where \( kprop: k \leq g n \) by blast
  with assms(3) dense_run_def obtain \( c \) where \( \text{hamlet}((\text{Rep}_\text{run} \text{sub}) k \ c) \)
  by blast
  moreover from \( kprop \) have \( \forall n. (g n < (\text{dil_inverse} f) n) \)
  unfolding not_less_iff_gr_or_eq by simp
  qed

8.1.5 Main Theorems

theory Stuttering
  imports StutteringLemmas
begin

Using the lemmas of the previous section about the invariance by stuttering of various prop-
erties of TESL specifications, we can now prove that the atomic formulae that compose TESL
specifications are invariant by stuttering.

Sporadic specifications are preserved in a dilated run.

lemma sporadic_sub:
  assumes \( \text{sub} \ll r \)
  and \( \text{sub} \in [c \text{ sporadic } \tau \text{ on } c']^\text{TESL} \)
  shows \( r \in [c \text{ sporadic } \tau \text{ on } c']^\text{TESL} \)
proof -
  from assms(1) is_subrun_def obtain \( f \)
  where \( \text{dilating} f \text{ sub } r \) by blast
  hence \( \forall n. \text{time}((\text{Rep}_\text{run} \text{sub}) n c) = \text{time}((\text{Rep}_\text{run} r) (f n) c) \wedge \text{hamlet}((\text{Rep}_\text{run} \text{sub}) n c) = \text{hamlet}((\text{Rep}_\text{run} r) (f n) c) \)
  by (simp add: dilating_def)
  moreover from assms(2) have \( \text{sub} \in [r. \exists n. \text{hamlet}((\text{Rep}_\text{run} r) n c) \wedge \text{time}((\text{Rep}_\text{run} r) n c') = \tau) \) by simp
8.1. STUTTERING INVARIANCE

from this obtain $k$ where $\langle \text{time ((Rep\_run \ sub) k c')} = \tau \land \text{hamlet ((Rep\_run \ sub) k c)} \rangle$ by auto ultimately have $\langle \text{time ((Rep\_run r) (f k) c')} = \tau \land \text{hamlet ((Rep\_run r) (f k) c)} \rangle$ by simp thus $\text{thesis}$ by auto

Implications are preserved in a dilated run.

theorem implies_sub:
assumes $\langle \text{sub} \triangleleft \text{r} \rangle$
and $\langle \text{sub} \in [\begin{array}{c}
\text{c}_1 \rightarrow \text{c}_2 \end{array}]_{\text{T\_ESL}} \rangle$
shows $\langle \text{r} \in [\begin{array}{c}
\text{c}_1 \rightarrow \text{c}_2 \end{array}]_{\text{T\_ESL}} \rangle$

proof -
from asms(1) is_subrun_def obtain $f$ where $\langle \text{dilating f sub r} \rangle$ by blast
moreover from asms(2) have
$\langle \text{sub} \in (\begin{array}{c}
\forall \ n. \text{hamlet ((Rep\_run r) n c}_1) \rightarrow \text{hamlet ((Rep\_run r) n c}_2) \end{array}) \rangle$ by simp
hence $\langle \forall \ n. \text{hamlet ((Rep\_run sub) n c}_1) \rightarrow \text{hamlet ((Rep\_run sub) n c}_2) \rangle$ by simp
ultimately have $\langle \forall \ n. \text{hamlet ((Rep\_run r) n c}_1) \rightarrow \text{hamlet ((Rep\_run r) n c}_2) \rangle$
using ticks_imp_ticks_subk ticks_sub by blast
thus $\text{thesis}$ by simp

qed

theorem implies_not_sub:
assumes $\langle \text{sub} \triangleleft \text{r} \rangle$
and $\langle \text{sub} \in [\begin{array}{c}
\text{c}_1 \rightarrow \neg \text{c}_2 \end{array}]_{\text{T\_ESL}} \rangle$
shows $\langle \text{r} \in [\begin{array}{c}
\text{c}_1 \rightarrow \neg \text{c}_2 \end{array}]_{\text{T\_ESL}} \rangle$

proof -
from asms(1) is_subrun_def obtain $f$ where $\langle \text{dilating f sub r} \rangle$ by blast
moreover from asms(2) have
$\langle \text{sub} \in (\begin{array}{c}
\forall \ n. \text{hamlet ((Rep\_run r) n c}_1) \rightarrow \neg \text{hamlet ((Rep\_run r) n c}_2) \end{array}) \rangle$ by simp
hence $\langle \forall \ n. \text{hamlet ((Rep\_run sub) n c}_1) \rightarrow \neg \text{hamlet ((Rep\_run sub) n c}_2) \rangle$ by simp
ultimately have $\langle \forall \ n. \text{hamlet ((Rep\_run r) n c}_1) \rightarrow \neg \text{hamlet ((Rep\_run r) n c}_2) \rangle$
using ticks_imp_ticks_subk ticks_sub by blast
thus $\text{thesis}$ by simp

qed

Precedence relations are preserved in a dilated run.

theorem weakly_precedes_sub:
assumes $\langle \text{sub} \triangleleft \text{r} \rangle$
and $\langle \text{sub} \in [\begin{array}{c}
\text{c}_1 \text{ weakly precedes } \text{c}_2 \end{array}]_{\text{T\_ESL}} \rangle$
shows $\langle \text{r} \in [\begin{array}{c}
\text{c}_1 \text{ weakly precedes } \text{c}_2 \end{array}]_{\text{T\_ESL}} \rangle$

proof -
from asms(1) is_subrun_def obtain $f$ where $\star: \langle \text{dilating f sub r} \rangle$ by blast
moreover from asms(2) have
$\langle \text{sub} \in (\begin{array}{c}
\forall \ n. \text{run\_tick\_count r c}_2 n \leq \text{run\_tick\_count r c}_1 n) \end{array}) \rangle$ by simp
hence $\langle \forall \ n. \text{run\_tick\_count sub c}_2 n \leq \text{run\_tick\_count sub c}_1 n) \rangle$ by simp
from dil_tick_count[OF asms(1) this]
have $\langle \forall \ n. \text{run\_tick\_count r c}_2 n \leq \text{run\_tick\_count r c}_1 n) \rangle$ by simp
thus $\text{thesis}$ by simp

qed

theorem strictly_precedes_sub:
assumes $\langle \text{sub} \triangleleft \text{r} \rangle$
and $\langle \text{sub} \in [\begin{array}{c}
\text{c}_1 \text{ strictly precedes } \text{c}_2 \end{array}]_{\text{T\_ESL}} \rangle$
shows $\langle \text{r} \in [\begin{array}{c}
\text{c}_1 \text{ strictly precedes } \text{c}_2 \end{array}]_{\text{T\_ESL}} \rangle$

proof -
from asms(1) is_subrun_def obtain $f$ where $\star: \langle \text{dilating f sub r} \rangle$ by blast
moreover from asms(2) have
$\langle \text{sub} \in (\begin{array}{c}
\forall n: \text{nat.} \text{run\_tick\_count g c}_2 n \leq \text{run\_tick\_count strictly g c}_1 n) \rangle$ by simp
with \(\text{strictly_precedes_alt_def2[of } (c_2) (c_1)]\) have
\[
\forall \text{sub} \in \{ \varrho, (\neg \text{hamlet } ((\text{Rep_run } \varrho) 0 c_2)) \}
\land (\forall n::\text{nat}. (\text{run_tick_count } \varrho c_2 (\text{Suc } n)) \leq (\text{run_tick_count } \varrho c_1 n))
\]
by blast

hence \((\neg \text{hamlet } ((\text{Rep_run } \text{sub}) 0 c_2))\)
\land (\forall n::\text{nat}. (\text{run_tick_count } c_2 (\text{Suc } n)) \leq (\text{run_tick_count } c_1 n))
by simp

hence
1:\((\neg \text{hamlet } ((\text{Rep_run } n)) 0 c_2))
\land (\forall n::\text{nat}. (\text{tick_count } c_2 (\text{Suc } n)) \leq (\text{tick_count } c_1 n))
by (simp add: tick_count_is_fun)

have (\forall n::\text{nat}. (\text{tick_count } r c_2 (\text{Suc } n)) \leq (\text{tick_count } r c_1 n))

proof -

\{ fix n::\text{nat} \\
\}

have (\text{tick_count } r c_2 (\text{Suc } n)) \leq (\text{tick_count } r c_1 n)

proof (cases \(\exists n_0, f n_0 = n\))

case True — \(n\) is in the image of \(f\)

from this obtain \(n_0\) where \(fn::f n_0 = n\) by blast

show ?thesis

proof (cases \(\exists n_0, f n_0 = \text{Suc } n\))

case True — \(\text{Suc } n\) is in the image of \(f\)

from this obtain \(n_0\) where \(\text{fsn}::f n_0 = \text{Suc } n\) by blast

using dilating_def dilating_fun_def by blast

with 1 have (\text{tick_count } \text{sub } c_2 n_0 \leq (\text{tick_count } \text{sub } c_1 n_0)) by simp

thus ?thesis using fn \(\text{fsn}::\text{tick_count}\_\text{sub}[\text{DF}*]\) \(\text{fn}\) by simp

next
case False — \(\text{Suc } n\) is not in the image of \(f\)

hence \((\neg \text{hamlet } ((\text{Rep_run } r) (\text{Suc } n)) c_2))\)

using \(\text{*}\) by (simp add: dilating_def dilating_fun_def)

hence (\text{tick_count } r c_2 (\text{Suc } n)) = (\text{tick_count } r c_2 n)

by (simp add: tick_countSuc)

also have (\(\ldots = \text{tick_count } \text{sub } c_2 n_0\))

using fn \(\text{tick_count}\_\text{sub}[\text{DF}*]\) by simp

finally have (\text{tick_count } r c_2 (\text{Suc } n)) = (\text{tick_count } \text{sub } c_2 n_0)

moreover have (\text{tick_count } \text{sub } c_2 n_0 \leq (\text{tick_count } \text{sub } c_2 \text{Suc } n_0))

by (simp add: tick_countSuc)

ultimately have (\text{tick_count } r c_2 (\text{Suc } n)) \leq (\text{tick_count } \text{sub } c_2 (\text{Suc } n_0))

by simp

moreover have (\text{tick_count } \text{sub } c_2 (\text{Suc } n_0)) \leq (\text{tick_count } \text{sub } c_1 n_0)

using \(\text{*}\) by simp

ultimately have (\text{tick_count } r c_2 (\text{Suc } n)) \leq (\text{tick_count } \text{sub } c_1 n_0)

by simp

thus ?thesis using tick_count_sub[DF*] \(\text{fn}\) by simp

qed

next
case False — \(n\) is not in the image of \(f\)

from greatest_prev_image[DF * this] obtain \(n_p\) where

\(\text{np_prop}::(\text{fn } n < n \land (\forall k. f n_p < k \land k \leq n \rightarrow (\exists k_0. f k_0 = k)))\) by blast

from tick_count_latest[DF * this] have
(\text{tick_count } r c_1 n = (\text{tick_count } r c_1 (f n_p)))

hence a: (\text{tick_count } r c_1 n = (\text{tick_count } \text{sub } c_1 n_p))

using tick_count_sub[DF*] \(\text{by simp}\)

have b: (\text{tick_count } \text{sub } c_2 (\text{Suc } n_p)) \leq (\text{tick_count } \text{sub } c_1 n_p)

using \(\text{I}\) by simp

show ?thesis

proof (cases \(\exists n_0, f n_0 = \text{Suc } n\))

case True — \(\text{Suc } n\) is in the image of \(f\)

from this obtain \(n_0\) where \(\text{fsn}::f n_0 = \text{Suc } n\) by blast

...
from \text{next\_non\_stuttering[OF } \ast \text{ np\_prop this]} \text{ have sn\_prop:}(sn_0 = \text{Suc } n_p) \).

with \( n \) have \((\text{tick\_count } \text{sub } c_2 \text{ sm}_0 \leq \text{tick\_count } \text{sub } c_1 \text{ n}_p) \) by simp

thus \( ?\text{thesis} \) using \( \text{tick\_count\_sub[OF } \ast \text{]} \) \( \text{fsn } a \) by auto

next

\text{case False} \quad \text{— Suc } n \text{ is not in the image of } f

hence \( \text{¬hamlet } ((\text{Rep\_run } r) (\text{Suc } n) c_2) \)

using \( \ast \) by (simp add: \( \text{dilating\_def} \) \( \text{dilating\_run\_def} \))

hence \((\text{tick\_count } r c_2 (\text{Suc } n) = \text{tick\_count } r c_2 n) \)

by (simp add: \( \text{tick\_count\_succ} \))

also have \((... = \text{tick\_count } c_2 n_p) \) using \( \text{np\_prop} \) \( \text{tick\_count\_sub[OF } \ast \text{]} \)

by (simp add: \( \text{tick\_count\_succ} \))

finally have \((\text{tick\_count } r c_2 (\text{Suc } n) = \text{tick\_count } \text{sub } c_2 n_p) \).

moreover have \((\text{tick\_count } \text{sub } c_2 n_p \leq \text{tick\_count } \text{sub } c_2 (\text{Suc } n_p)) \)

by (simp add: \( \text{tick\_count\_succ} \))

ultimately have \((\text{tick\_count } r c_2 (\text{Suc } n) \leq \text{tick\_count } \text{sub } c_2 (\text{Suc } n_p)) \) by simp

moreover have \((\text{tick\_count } \text{sub } c_2 (\text{Suc } n_p) \leq \text{tick\_count } \text{sub } c_1 n_p) \) using \( \ast \) by simp

ultimately have \((\text{tick\_count } r c_2 (\text{Suc } n) \leq \text{tick\_count } \text{sub } c_1 n_p) \) by simp

thus \( ?\text{thesis} \) using \( \text{np\_prop} \) \( \text{mono\_tick\_count} \) \( \text{using} \) \( \text{a} \) by \( \text{linarith} \)

qed

moreover from \( \ast \) have \((\text{¬hamlet } ((\text{Rep\_run } r) 0 c_2)) \)

using \( \ast \) \( \text{empty\_dilated\_prefix} \) \( \text{ticks\_sub} \) by \( \text{fastforce} \)

ultimately show \( ?\text{thesis} \) by (simp add: \( \text{tick\_count\_is\_run} \) \( \text{strictly\_precedes\_alt2} \))

\text{qed}

\text{Time\ delayed\ relations\ are\ preserved\ in\ a\ dilated\ run.}

\text{ theorem time\_delayed\_sub:}

\text{ assumes } (\text{sub } \ll \text{r})

\text{ and } (\text{sub } \subseteq [\text{a } \text{time\-delayed\ by } \delta \tau\ \text{on } ms \text{ implies } b]_{T_E S L})

\text{ shows } (r \subseteq [\text{a } \text{time\-delayed\ by } \delta \tau\ \text{on } ms \text{ implies } b]_{T_E S L})

\text{ proof -}

\text{ from \text{assms}(1) \text{is\_subrun\_def} \text{obtain} } f\ \text{where } \ast:\text{dilating } f\ \text{sub } r\ \text{by \text{blast}}

\text{ from \text{assms}(2) \text{have } \forall n. \text{hamlet } ((\text{Rep\_run } r) n a)\}

\text{ —— } ((\forall m \geq n. \text{first\_time } ms m (\text{time } ((\text{Rep\_run } r) n ms) + \delta \tau)) \text{ hamlet } ((\text{Rep\_run } r) m b))

\text{ using \text{T\_ESL}\_interpretation\_atomic\_simp\_sims(5)[of } (a) \langle \delta \tau\rangle (ms) (b)\rangle \text{ by \text{simp}}

\text{ hence } **:\forall n_0. \text{hamlet } ((\text{Rep\_run } r) (f n_0) a)\}

\text{ —— } ((\forall n_0 \geq n_0. \text{first\_time } ms (f n_0) \text{ (time } ((\text{Rep\_run } r) (f n_0) ms) + \delta \tau)) \text{ hamlet } ((\text{Rep\_run } r) (f n_0) b)) \}

\text{ using \text{first\_time\_image[OF } \ast \text{]} \text{dilating\_def} \ast \text{by \text{fastforce}}

\text{ hence } \forall n. \text{hamlet } ((\text{Rep\_run } r) n a)\}

\text{ —— } ((\forall n \geq n. \text{first\_time } ms (f (n a) \text{ (time } ((\text{Rep\_run } r) n ms) + \delta \tau)) \text{ hamlet } ((\text{Rep\_run } r) m b))\}

\text{ proof -}

\{ \text{ fix } n \text{ assume } \text{assm:} \text{hamlet } ((\text{Rep\_run } r) n a)\}

\text{ from \text{ticks\_image\_sub[OF } \ast \text{ assm]} \text{obtain } n_0\ \text{where mnf0:}(n = f n_0) \text{ by \text{blast}}

\text{ with } **\text{ have } ft0:\}

\text{ ((\forall n_0 \geq n_0. \text{first\_time } ms (f n_0) \text{ (time } ((\text{Rep\_run } r) (f n_0) ms) + \delta \tau)) \text{ hamlet } ((\text{Rep\_run } r) (f n_0) b)) \text{ by \text{blast}}

\text{ have } ((\forall n \geq n. \text{first\_time } ms (f (n ms) + \delta \tau)) \text{ hamlet } ((\text{Rep\_run } r) m b)) \text{ \text{ by \text{fastforce}}}

\text{ proof -}

\{ \text{ fix } n \text{ assume } \text{hyp:}(n \geq n)\}

\text{ have } ((\forall n ms (f (n ms) + \delta \tau)) \text{ hamlet } ((\text{Rep\_run } r) m b))\)
CHAPTER 8. PROPERTIES OF TESL

proof (cases \( \exists m_0. \, f m_0 = m \))
  case True
  from this obtain \( m_0 \) where \( m = f m_0 \) by blast
  moreover have \( \text{strict_mono } f \) using \( * \) by (simp add: dilating_def dilating_fun_def)
  ultimately show \( \text{thesis} \) using \( ft0 \) hyp \( nfn0 \) by (simp add: strict_mono_less_eq)
next
  case False thus \( \text{thesis} \) proof (cases \( m = 0 \))
    case True
    hence \( \exists pm. \, m = Suc pm \) by (simp add: not0_implies_Suc)
    from this obtain \( pm \) where \( \langle m = Suc pm \rangle \) using \( * \) by simp
    hence \( \forall c. \, \text{time (Rep_run r (Suc pm) c)} = \text{time (Rep_run r n c)} \) by blast
    moreover have \( \text{thesis} \) using \( \text{thesis} \) by blast
    moreover have \( \forall c. \, \text{time (Rep_run r (Suc pm) c)} = \text{time (Rep_run r n c)} \) by blast
    hence \( \text{thesis} \) using \( \text{thesis} \) by blast
  next
    case False
    hence \( \forall c. \, \text{time (Rep_run r (Suc pm) c)} = \text{time (Rep_run r n c)} \) by blast
    hence \( \text{thesis} \) using \( \text{thesis} \) by blast
  qed
qed
}
thus \( \text{thesis} \) by simp
qed

time relations are preserved through dilation of a run.

lemma tagrel_sub':
  assumes \( \text{sub} \prec r \)
  and \( \text{sub} \in \{ \text{time-relation } [c_1, c_2] \in \text{R_TESL} \} \)
  shows \( \forall c. \, \text{time (Rep_run r n c)} = \text{time (Rep_run sub n c)} \)
proof
  from assms(1) is_subrun_def obtain \( f \) where \( \text{dilating } f \) using \( * \) by blast
  moreover from assms(2) TESL_interpretation_atomic.simps(2) have
  \( \forall c. \, \text{time (Rep_run sub n c)} = \text{time (Rep_run r n c)} \)
  hence \( \text{thesis} \) using \( \text{thesis} \) by blast
  qed
qed

lemma tagrel_sub:
  assumes \( \forall c. \, \text{time (Rep_run r n c)} = \text{time (Rep_run sub n c)} \)
  shows \( \forall c. \, \text{time (Rep_run r n c)} = \text{time (Rep_run sub n c)} \)
proof
  (induction n)
next
  case (Suc n)
  hence \( \forall c. \, \text{time (Rep_run r n c)} = \text{time (Rep_run sub n c)} \)
  by (simp add: dilating_def dilating_fun_def)
8.1. STUTTERING INVARIANCE

thus ?thesis using Suc.INH by simp
next
case False
from this obtain n0 where n0prop: (f n0 = Suc n) by blast
from 1 have ?thesis (time ((Rep_run sub) n0 c1), time ((Rep_run sub) n0 c2)) by simp
moreover from n0prop have time ((Rep_run sub) n0 c1) = time ((Rep_run r) (Suc n) c1)
  by (simp add: dilating_def) 
moreover from n0prop have time ((Rep_run sub) n0 c2) = time ((Rep_run r) (Suc n) c2)
  by (simp add: dilating_def) 
ultimately show ?thesis by simp
qed

{ fix x assume \[ \text{hamlet (Rep}_{0}\text{run r (f n) c1)} \]
  with 1 have 2: \[ \forall m \geq n. \neg \text{hamlet (Rep}_{0}\text{run r (f m) c2)} \] by simp
  have \[ \forall n \geq (f n). \neg \text{hamlet (Rep}_{0}\text{run r n c2)} \]
  proof (cases \[ \exists m. f n = m \] )
  have (\neg \text{hamlet (Rep}_{0}\text{run r n c2)})
  proof (cases \[ \exists m. f n = m \] )
chapter 8. Properties of TESL

```plaintext
case True
  from this obtain m_0 where fm0: m_0 = m_0 by blast
  hence m_0 ≥ n
  using * dilating_def dilating_fun_def h strict_mono_less_eq by fastforce
  with 2 show thesis using fm0 by blast
next
  case False
  thus thesis using TESL_interpretation_atomic.simps(8) by blast
qed

lemmas atomic_sub_lemmas = sporadic_sub tagrel_sub implies_sub implies_not_sub
  time_delayed_sub weakly_precedes_sub
  strictly_precedes_sub kill_sub

We can now prove that all atomic specification formulae are preserved by the dilation of runs.

lemma atomic_sub:
  assumes ⟨sub ≪ r⟩ and ⟨sub ∈ ⌈ϕ⌉ TESL⟩
  shows ⟨r ∈ ⌈ϕ⌉ TESL⟩
  using assms(2) atomic_sub_lemmas[OF assms(1)] by (cases ϕ, simp_all)

Finally, any TESL specification is invariant by stuttering.

theorem TESL_stuttering_invariant:
  assumes ⟨sub ≪ r⟩
  shows σ ∈ ⌈S⌉ TESL ⇒ r ∈ ⌈S⌉ TESL
  proof (induction S)
    case Nil
    thus case by simp
next
  case (Cons a s)
    from Cons.prems have sa: ⟨sub ∈ ⌈a⌉ TESL⟩ and sb: ⟨sub ∈ ⌈s⌉ TESL⟩
      using TESL_interpretation_image by simp
    from Cons.IH[OF sb] have ‘r ∈ ⌈s⌉ TESL’.
      moreover from atomic_sub[OF assms(1)] sa have ‘r ∈ ⌈a⌉ TESL’.
      ultimately show thesis using TESL_interpretation_image by simp
    qed

end
theory Config_Morphisms
  imports Hygge_Theory
begin

TESL morphisms change the time on clocks, preserving the ticks.

consts morphism :: ’a ⇒ (’τ::linorder ⇒ ’τ::linorder) ⇒ ’a ⇒ (infxs1 (⊗) 100)

Applying a TESL morphism to a tag simply changes its value.

overloading morphism_tagconst ≡ morphism :: ’τ tag_const ⇒ (’τ::linorder ⇒ ’r) ⇒ ’τ tag_const

begin
```

end
8.1. STUTTERING INVARIANCE

definition morphism_tagconst :
  \((x::'\tau \text{ tag const}) \otimes (f::('\tau::\text{linorder} \Rightarrow '\tau)) = (\tau_{\text{cst}} \circ f)(\text{the_tag_const x})\)
end

Applying a TESL morphism to an atomic formula only changes the dates.

overloading morphism_TESL_atomic 
≡ morphism :: 'τ TESL_atomic ⇒ ('τ::linorder ⇒ 'τ) ⇒ 'τ TESL_atomic

begin
  definition morphism_TESL_atomic :
    \((\Psi::'τ TESL_atomic) \otimes (f::('τ::linorder ⇒ 'τ)) = \text{map}(\lambda x. x \otimes f)\Psi\)
end

Applying a TESL morphism to a formula amounts to apply it to each atomic formula.

overloading morphism_TESL_formula 
≡ morphism :: 'τ TESL_formula ⇒ ('τ::linorder ⇒ 'τ) ⇒ 'τ TESL_formula

begin
  definition morphism_TESL_formula :
    \((\Psi::'τ TESL_formula) \otimes (f::('τ::linorder ⇒ 'τ)) = \text{map}(\lambda x. x \otimes f)\Psi\)
end

Applying a TESL morphism to a configuration amounts to apply it to the present and future formulae. The past (in the context Γ) is not changed.

overloading morphism_TESL_config 
≡ morphism :: ('τ::\text{linordered_field}) config ⇒ ('τ ⇒ 'τ) ⇒ 'τ config

begin
  fun morphism_TESL_config 
  where
    \(((\Gamma, n \vdash \Psi \triangleright \Phi)::('\tau::\text{linordered_field}) \text{config}) \otimes (f::('\tau ⇒ 'τ)) = (\Gamma, n \vdash (\Psi \otimes f) \triangleright (\Phi \otimes f))\)
end

A TESL formula is called consistent if it possesses Kripke-models in its denotational interpretation.

definition consistent :: ('τ::\text{linordered_field}) TESL_formula ⇒ bool

where
  \((\text{consistent } \Psi \equiv [[\Psi ]]_{TESL} \neq \emptyset)\)

If we can derive a consistent finite context from a TESL formula, the formula is consistent.

theorem consistency_finite :
  assumes start : \(((\emptyset, 0 \vdash \Psi \triangleright \emptyset) \Rightarrow (\Gamma_1, n_1 \vdash \emptyset \triangleright \emptyset))\)
  and init_invariant : (consistent_context \Gamma_1)
  shows (consistent \Psi)

proof
  have \(* : \exists n. ((\emptyset, 0 \vdash \Psi \triangleright \emptyset) \Rightarrow^* (\Gamma_1, n_1 \vdash \emptyset \triangleright \emptyset))\)
  by (simp add: rtranclp_imp_relpowp start)
  show ?thesis 
  unfolding consistent_context_def consistent_def 
  using * consistent_context_def init_invariant soundness by fastforce
qed
Snippets on runs

A run with no ticks and constant time for all clocks.

**Definition**

\[
\text{const\_nontick\_run} : \langle (\text{clock} \Rightarrow \tau \text{ tag\_const}) \Rightarrow (\tau :: \text{linordered\_field}) \text{ run} \rangle (\square) 80
\]

where \( \square f \equiv \text{Abs\_run}(\lambda n c. (\text{False}, f c)) \)

Ensure a clock ticks in a run at a given instant.

**Definition**

\[
\text{set\_tick} : \langle (\tau :: \text{linordered\_field}) \text{ run} \Rightarrow \text{nat} \Rightarrow \text{clock} \Rightarrow (\tau) \text{ run} \rangle
\]

where \( \text{set\_tick} r k c = \text{Abs\_run}(\lambda n c. \text{if } k = n \text{ then } (\text{True}, \text{time}(\text{Rep\_run} r k c)) \text{ else } \text{Rep\_run} r k c) \)

Ensure a clock does not tick in a run at a given instant.

**Definition**

\[
\text{unset\_tick} : \langle (\tau :: \text{linordered\_field}) \text{ run} \Rightarrow \text{nat} \Rightarrow \text{clock} \Rightarrow (\tau) \text{ run} \rangle
\]

where \( \text{unset\_tick} r k c = \text{Abs\_run}(\lambda n c. \text{if } k = n \text{ then } (\text{False}, \text{time}(\text{Rep\_run} r k c)) \text{ else } \text{Rep\_run} r k c) \)

Replace all instants after \( k \) in a run with the instants from another run. Warning: the result may not be a proper run since time may not be monotonous from instant \( k \) to instant \( k+1 \).

**Definition**

\[
\text{patch} : \langle (\tau :: \text{linordered\_field}) \text{ run} \Rightarrow \text{nat} \Rightarrow (\tau) \text{ run} \Rightarrow (\tau) \text{ run} \rangle (\rightarrow) 80
\]

where \( \text{patch} r k r' \equiv \text{Abs\_run}(\lambda n c. \text{if } n \leq k \text{ then } \text{Rep\_run} (r) n c \text{ else } \text{Rep\_run} (r') n c) \)

For some infinite cases, the idea for a proof scheme looks as follows: if we can derive from the initial configuration \( \emptyset, 0 \vdash \Psi > \emptyset \) a start-point of a lasso \( \Gamma_1, n_1 \vdash \Psi_1 > \Phi_1 \), and if we can traverse the lasso one time \( \Gamma_1, n_1 \vdash \Psi_1 > \Phi_1 \leftrightarrow ++ \Gamma_2, n_2 \vdash \Psi_2 > \Phi_2 \) to isomorphic one, we can always always make a derivation along the lasso. If the entry point of the lasso had traces with prefixes consistent to \( \Gamma_1 \), then there exist traces consisting of this prefix and repetitions of the delta-prefix of the lasso which are consistent with \( \Psi \) which implies the logical consistency of \( \Psi \).

So far the idea. Remains to prove it. Why does one symbolic run along a lasso generalises to arbitrary runs?

**Theorem**

\[ \text{consistency\_coinduct} : \]

\[ \text{assumes start : } (\emptyset, 0 \vdash \Psi > \emptyset) \leftrightarrow (\Gamma_1, n_1 \vdash \Psi_1 > \Phi_1) \]

\[ \text{and loop : } (\Gamma_1, n_1 \vdash \Psi_1 > \Phi_1) \leftrightarrow ++ (\Gamma_2, n_2 \vdash \Psi_2 > \Phi_2) \]

\[ \text{and init\_invariant : } (\text{consistent\_context } \Gamma_1) \]

\[ \text{and post\_invariant : } (\text{consistent\_context } \Gamma_2) \]

\[ \text{and retract\_condition : } (\Gamma_2, n_2 \vdash \Psi_2 > \Phi_2) \otimes (f :: \tau \Rightarrow \tau) = (\Gamma_1, n_1 \vdash \Psi_1 > \Phi_1) \]

\[ \text{shows } (\text{consistent } (\Psi :: (\tau :: \text{linordered\_field}) \text{TESL\_formula})) \]

oops
Bibliography
