A Formal Development of a Polychronous Polytimed Coordination Language

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

Frédéric Boulanger
frereric.boulanger@centralesupelec.fr

Burkhart Wolff
burkhart.wolff@lri.fr

April 20, 2020
# Contents

1 A Gentle Introduction to TESL 5

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

7 Main Theorems 47
  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 63
  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
CHAPTER 1. A GENTLE INTRODUCTION TO TESL

Figure 1.1: A Heterogeneous Timed System Model

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: $\Sigma^\infty = \mathbb{N} \rightarrow \mathcal{K} \rightarrow (\mathbb{B} \times \mathcal{T})$, 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 $\mathcal{T}$ 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.

- c1 implies c2 means that at any instant where c1 ticks, c2 has to tick too.
- c1 implies not c2 means that at any instant where c1 ticks, c2 cannot tick.
- c1 kills c2 means that at any instant where c1 ticks, and at any future instant, c2 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.
- c1 sporadic t on c2 means that clock c1 must have a tick at an instant where the time on c2 is t.
- c1 time delayed by d on m implies c2 means that every time clock c1 ticks, c2 must have a tick at the first instant where the time on m is d later than it was when c1 had ticked. This means that every tick on c1 is followed by a tick on c2 after a delay d measured on the time scale of clock m.
- time relation (c1, c2) in R means that at every instant, the current time on clocks c1 and c2 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.

- c1 weakly precedes c2 means that for each tick on c2, there must be at least one tick on c1 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 c2 than on c1.

\(^1\)See http://wdi.supelec.fr/software/TESL/GalleryEngine for more details
• $c_1$ strictly precedes $c_2$ means that for each tick on $c_2$, there must be at least one tick on $c_1$ at a previous instant. This can also be expressed by saying that at each instant, the number of ticks on $c_2$ from the beginning of the run to this instant, must be lower or equal to the number of ticks on $c_1$ 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_const⟩)

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

type synonym \( \tau \) TESL_formula = \( \langle \tau \) 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 :: \( \langle \tau \) TESL_atomic \Rightarrow \text{bool} \) where
\( \langle \text{positive_atom} (_ \text{sporadic} _) \text{on} _ = \text{True} \rangle \)
\( \langle \text{positive_atom} _ = \text{False} \rangle \)

The NoSporadic function removes sporadic constraints from a TESL formula.

abbreviation NoSporadic :: \( \langle \tau \) TESL_formula \Rightarrow \tau \) TESL_formula \)
where
\( \text{NoSporadic} f \equiv \text{List.filter} (\lambda f \text{atom}. \text{case} f \text{atom} \text{of} \)
\( \langle _ \text{sporadic} _ \text{on} _ \Rightarrow \text{False} \rangle \)
\( \langle _ \Rightarrow \text{True} \rangle \text{f} \))

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 \( \tau \text{tag_const} \) is itself a field.

instantiation tag_const ::(field)field
begin
  fun inverse_tag_const
  where (inverse \( \tau \text{est} t \)) = \( \tau \text{est} \) (inverse t)

  fun divide_tag_const
  where (divide \( \tau \text{est} t_1 \) \( \tau \text{est} t_2 \)) = \( \tau \text{est} \) (divide t_1 t_2)

  fun uminus_tag_const
  where (uminus \( \tau \text{est} t \)) = \( \tau \text{est} \) (uminus t)

  fun minus_tag_const
  where (minus \( \tau \text{est} t_1 \) \( \tau \text{est} t_2 \)) = \( \tau \text{est} \) (minus t_1 t_2)

  definition (one_tag_const \equiv \tau \text{est} 1)

  fun times_tag_const
  where (times \( \tau \text{est} t_1 \) \( \tau \text{est} t_2 \)) = \( \tau \text{est} \) (times t_1 t_2)

  definition (zero_tag_const \equiv \tau \text{est} 0)

  fun plus_tag_const
  where (plus \( \tau \text{est} t_1 \) \( \tau \text{est} t_2 \)) = \( \tau \text{est} \) (plus t_1 t_2)

instance proof

fix a::\( \langle \tau ::\text{field tag_const} \rangle \) and b::\( \langle \tau ::\text{field tag_const} \rangle \)
and c::\( \langle \tau ::\text{field tag_const} \rangle \)

obtain u v w where \( a = \tau \text{est} u \) and \( b = \tau \text{est} v \) and \( c = \tau \text{est} w \)
using tag_const.exhaust by metis
thus \((a \cdot b \cdot c = a \cdot (b \cdot c))\)

by (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 tag_const.exhaust by metis

thus \((a \cdot b = b \cdot a)\)

by (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 tag_const.exhaust by blast

thus \((1 \cdot a = a)\)

by (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 tag_const.exhaust by metis

thus \((a + b + c = a + (b + c))\)

by (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 tag_const.exhaust by metis

thus \((a + b = b + a)\)

by (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 tag_const.exhaust by blast

thus \((0 + a = a)\)

by (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 tag_const.exhaust by blast

thus \((-a + a = 0)\)

by (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 tag_const.exhaust by metis

thus \((a - b = a + -b)\)

by (simp add: TESL.minus_tag_const.simps)
next

Distributive property of multiplication over addition.

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
thus ⟨(a + b) * c = a * c + b * c⟩
by (simp add: TESL.plus_tag_const.simps
    TESL.times_tag_const.simps
    ring_class.ring_distribs(2))

next

The neutral elements are distinct.

show ⟨(0::('τ::field tag const)) ≠ 1⟩
by (simp add: one_tag_const_def zero_tag_const_def)
next

The product of an element and its inverse is 1.

fix a ::= ('τ::field tag const)
assume h: ⟨a ≠ 0⟩
obtain u where ⟨a = τ_cst u⟩ using tag_const.exhaust by blast
moreover with h have ⟨u ≠ 0⟩ by (simp add: zero_tag_const_def)
ultimately show ⟨inverse a * a = 1⟩
by (simp add: TESL.inverse_tag_const.simps
    TESL.times_tag_const.simps
    one_tag_const_def)
next

Dividing is multiplying by the inverse.

fix a ::= ('τ::field tag const) and b ::= ('τ::field tag const)
obtain u v where ⟨a = τ_cst u⟩ and ⟨b = τ_cst v⟩ using tag_const.exhaust by metis
thus ⟨a div b = a * inverse b⟩
by (simp add: TESL.divide_tag_const.simps
    TESL.inverse_tag_const.simps
    TESL.times_tag_const.simps
    divide_inverse)
next

Zero is its own inverse.

show ⟨inverse (0::('τ::field tag const)) = 0⟩
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.

instantiation tag_const :: (order)order
begin
inductive less_eq_tag_const :: ('a tag const ⇒ 'a tag const ⇒ bool)
where
  Int_less_eq[simp]: ⟨n ≤ m⟩ ⇒ (TConst n) ≤ (TConst m)
definition less_tag: ⟨(x::'a tag const) < y ⟷ (x ≤ y) ∧ (x ≠ y)⟩
2.2. DEFINING RUNS

instance proof
  show \(\forall x \leq y \leq z. x \leq z \Rightarrow y \leq z \Rightarrow x \leq z\)
    using less_eq_tag_const.simps by (auto)
  next
    show \(\forall x \leq y \leq x. y \leq x \Rightarrow x = y\)
    using less_eq_tag_const.simps by (auto)
  qed

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 tag_const) and y::("a tag_const)
    from tag_const.exhaust obtain x0::"a where (x = TConst x0) by blast
    moreover from tag_const.exhaust obtain y0::"a where (y = TConst y0) by blast
    ultimately show \(\forall x \leq y \vee y \leq x\) 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 't instant = (clock \Rightarrow ('s t instant \
\times \'s 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) 't::linordered_field run = \('\rho::\text{nat} \Rightarrow \tau\ instant. \forall c. \text{mono} (\lambda n. \text{time} (\rho n c))\)
proof
  show \((\lambda _. \ (\text{True}, \tau \text{const}, 0)) \in \{\rho. \forall c. \text{mono} (\lambda n. \text{time} (\rho n c))\)\)
  unfolding mono_def by blast
qed
A *dense* run is a run in which something happens (at least one clock ticks) at every instant.

**Definition**:

\[
\text{dense_run} \varphi \equiv (\forall n. \exists c. \text{hamlet} (\text{Rep_run} \varphi n c))
\]

**fun run_tick_count**: \((\tau::\text{linordered_field}) \text{ run} \Rightarrow \text{clock} \Rightarrow \text{nat} \Rightarrow \text{nat}\)

where

\[
\begin{align*}
\text{run_tick_count} \varphi K 0 & = (\text{if hamlet} (\text{Rep_run} \varphi 0 K) \text{ then } 1 \text{ else } 0) \\
\text{run_tick_count} \varphi K (\text{Suc } n) & = (\text{if hamlet} (\text{Rep_run} \varphi (\text{Suc } n) K) \text{ then } 1 + (\text{run_tick_count} \varphi K n) \text{ else } (\text{run_tick_count} \varphi K n))
\end{align*}
\]

**fun run_tick_count_strictly**: \((\tau::\text{linordered_field}) \text{ run} \Rightarrow \text{clock} \Rightarrow \text{nat} \Rightarrow \text{nat}\)

where

\[
\begin{align*}
\text{run_tick_count_strictly} \varphi K 0 & = 0 \\
\text{run_tick_count_strictly} \varphi K (\text{Suc } n) & = \text{run_tick_count} \varphi K n
\end{align*}
\]

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

**Definition**:

\[
\text{first_time} \equiv (\text{time} (\text{Rep_run} \varphi n K) = \tau) \land (\forall n'. n' < n \land \text{time} (\text{Rep_run} \varphi n' K) = \tau))
\]

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

**Lemma before_first_time**: assumes \((\forall n. \text{time} (\text{Rep_run} \varphi n K) < \tau)\)

and \((m < n)\) shows \((\text{time} (\text{Rep_run} \varphi m K) < \tau)\)

**Proof** -

- have \((\text{mono } (\lambda n. \text{time} (\text{Rep_run} \varphi n K)))\) using \text{Rep_run} by blast
- moreover from \text{assms}(2) have \((m \leq n)\) using \text{less_imp_le} by simp
- moreover have \((\text{mono } (\lambda n. \text{time} (\text{Rep_run} \varphi n K)))\) using \text{Rep_run} by blast
- ultimately have \((\text{time} (\text{Rep_run} \varphi m K) \leq \text{time} (\text{Rep_run} \varphi n K))\)
- by \(\text{simp add: mono_def}\)
- moreover from \text{assms}(1) have \((\text{time} (\text{Rep_run} \varphi n K) = \tau)\)
- using \text{first_time_def} by blast
- moreover from \text{assms} have \((\text{time} (\text{Rep_run} \varphi m K) \neq \tau)\)
- using \text{first_time_def} by blast
- ultimately show \(?\text{thesis by simp}\)

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

**Lemma alt_first_time_def**: assumes \((\forall n. \text{time} (\text{Rep_run} \varphi n K) < \tau)\)
2.2. DEFINING RUNS

and \((\text{time } ((\text{Rep\_run } \varrho) \ n \ K) = \tau)\)
shows \(\langle \text{first\_time } \varrho \ K \ n \tau \rangle\)
proof -
from assms(1) have \(\langle \forall n < n. \text{time } ((\text{Rep\_run } \varrho) \ n \ K) \neq \tau \rangle\)
  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 = (\\exists n::nat. hamlet ((Rep_run ϱ) n K₁) ∧ time ((Rep_run ϱ) 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 = (\\forall n::nat. R (time ((Rep_run ϱ) n K₁), time ((Rep_run ϱ) n K₂)))
— master implies slave means that at each instant at which master ticks, slave also ticks.

⟦master implies slave⟧TESL = (\\forall n::nat. hamlet ((Rep_run ϱ) n master) → hamlet ((Rep_run ϱ) n slave))
— master implies not slave means that at each instant at which master ticks, slave does not tick.

⟦master implies not slave⟧TESL = (\\forall n::nat. hamlet ((Rep_run ϱ) n master) → ¬hamlet ((Rep_run ϱ) 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 = (\\forall m ≥ n. first_time measuring m (measured_time + δτ))
→ hamlet ((Rep_run ϱ) m slave)

K₁ weakly precedes K₂ means that each tick on K₂ must be preceded by or coincide with at least one tick on K₁. Therefore, at each instant n, the number of ticks on K₂ must be less or equal to the number of ticks on K₁.

| ⟨ ⟨ K₁ weakly precedes K₂ ⟩ ⟩ TESL = {ϱ. ∀n::nat. (run_tick_count ϱ K₂ n) ≤ (run_tick_count ϱ K₁ n)} |
| ⟨ ⟨ K₁ strictly precedes K₂ ⟩ ⟩ TESL = {ϱ. ∀n::nat. (run_tick_count ϱ K₂ n) ≤ (run_tick_count_strictly ϱ K₁ n)} |
| ⟨ ⟨ K₁ kills K₂ ⟩ ⟩ TESL = {ϱ. ∀n::nat. hamlet ((Rep_run ϱ) n K₁) → (∀m≥n. ¬hamlet ((Rep_run ϱ) m K₂))} |

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.

fun TESL_interpretation :: ('τ::linordered_field) TESL_formula ⇒ 'τ run set
where
⟨ ⟨ [] ⟩ ⟩ TESL = {_. True}
⟨ ⟨ ϕ # Φ ⟩ ⟩ TESL = [ ϕ ] TESL ∩ [ [ Φ ] ] TESL |
lemma TESL_interpretation_homo:
by simp

3.2.1 Image interpretation lemma

theorem TESL_interpretation_image:
by (induction Φ, simp+)

3.2.2 Expansion law

Similar to the expansion laws of lattices.

theorem TESL_interp_homo_append:
by (induction Φ₁, simp, auto)

3.3 Equational laws for the denotation of TESL formulae

lemma TESL_interp_assoc:
[ [ (Φ₁ @ Φ₂) @ Φ₃ ] ] TESL = [ [ Φ₁ @ (Φ₂ @ Φ₃) ] ] TESL |
by auto

lemma TESL_interp_commute:
shows [ [ Φ₁ @ Φ₂ ] ] TESL = [ [ Φ₂ @ Φ₁ ] ] TESL |
by (simp add: TESL_interp_homo_append inf_sup_asc(1))
3.4 DECREASING INTERPRETATION OF TESL FORMULAE

lemma TESL_interp_left_commute:
\[
[[\Phi_1 \circ (\Phi_2 \circ \Phi_3)]_{TESL}] = [[\Phi_2 \circ (\Phi_1 \circ \Phi_3)]_{TESL}}
\]
unfolding TESL_interp_homo_append by auto

lemma TESL_interp_idem:
\[
[[\Phi \circ \Phi]_{TESL}] = [[\Phi]_{TESL}}
\]
using TESL_interp_homo_append by auto

lemma TESL_interp_left_idem:
\[
[[\Phi_1 \circ (\Phi_1 \circ \Phi_2)]_{TESL}] = [[\Phi_1 \circ \Phi_2]_{TESL}}
\]
using TESL_interp_homo_append by auto

lemmas TESL_interp_aci = TESL_interp_commute
TESL_interp_assoc
TESL_interp_left_commute
TESL_interp_left_idem

The empty formula is the identity element.

lemma TESL_interp_neutral1:
\[
[[\emptyset \circ \Phi]_{TESL}] = [[\Phi]_{TESL}}
\]
by simp

lemma TESL_interp_neutral2:
\[
[[\Phi \circ \emptyset]_{TESL}] = [[\Phi]_{TESL}}
\]
by simp

3.4 Decreasing interpretation of TESL formulae

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

lemma TESL_sem_decreases_head:
\[
[[\Phi]_{TESL}] \supseteq [[\Phi \# \emptyset]_{TESL}}
\]
by simp

lemma TESL_sem_decreases_tail:
\[
[[\Phi \# \emptyset]_{TESL}] \supseteq [[\Phi \circ [\emptyset]_{TESL}}
\]
by (simp add: TESL_interp_homo_append)

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

lemma TESL_interp_formula_stuttering:
assumes \( \emptyset \in \text{set}\ \Phi \)
shows \( [[\emptyset \# \Phi]_{TESL}] = [[\Phi]_{TESL}} \)
proof -
have \( \emptyset \# \Phi = [\emptyset] \circ \Phi \) by simp
hence \( [[\emptyset \# \Phi]_{TESL}] = [[\emptyset]_{TESL} \circ [[\emptyset]_{TESL}} \)
using TESL_interp_homo_append by simp
thus \( \text{thesis} \) using assms TESL_interpretation_image by fastforce
qed

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

lemma TESL_interp_remdups_absorb:
\[
[[\Phi]_{TESL}] = [[\text{remdups} \\Phi]_{TESL}}
\]
proof (induction \( \Phi \))
    case Cons
    thus \( \exists \Phi \) 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 \( \Phi \) = set \( \Phi' \))
    shows (\( \llbracket \Phi \rrbracket_{TESL} \supseteq \llbracket \Phi' \rrbracket_{TESL} \))
proof -
    have (set (rendups \( \Phi \)) = set (rendups \( \Phi' \)))
        by (simp add: assms)
    moreover have \( \exists \Phi \) where decompose: (set (\( \Phi \sqcap \Phi' \)) = set \( \Phi' \)) using assms by auto
    hence \( \exists \Phi \) where decompose: (set (\( \Phi \sqcap \Phi' \)) = set \( \Phi' \)) using assms by blast
    moreover have \( \exists \Phi' \) where decompose: (set (\( \Phi \cup \Phi' \)) = set \( \Phi' \))
        by (simp add: TESL_interp_set_lifting decompose)
    moreover have \( \llbracket \Phi' \rrbracket_{TESL} = \llbracket \Phi \sqcap \Phi' \rrbracket_{TESL} \)
        using TESL_interp_set_lifting decompose by blast
    moreover have \( \llbracket \Phi \sqcap \Phi' \rrbracket_{TESL} \supseteq \llbracket \Phi \rrbracket_{TESL} \cap \llbracket \Phi' \rrbracket_{TESL} \)
        by (simp add: TESL_interp_homo_append)
    moreover have \( \llbracket \Phi \rrbracket_{TESL} \supseteq \llbracket \Phi \rrbracket_{TESL} \cap \llbracket \Phi' \rrbracket_{TESL} \) by simp
    ultimately show \( \exists \Phi \) using TESL_interp_remdups_absorb by auto
qed

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

theorem TESL_interp_decreases_setinc:
    assumes \( \Phi \subseteq \Phi' \)
    shows \( \llbracket \Phi \rrbracket_{TESL} \supseteq \llbracket \Phi' \rrbracket_{TESL} \)
proof -
    obtain \( \Phi' \) where decompose: (set (\( \Phi \sqcap \Phi' \)) = set \( \Phi' \)) using assms by auto
    hence \( \exists \Phi \) where decompose: (set (\( \Phi \sqcap \Phi' \)) = set \( \Phi' \)) using assms by blast
    moreover have \( \exists \Phi' \) where decompose: (set (\( \Phi \cup \Phi' \)) = set \( \Phi' \))
        by (simp add: TESL_interp_set_lifting decompose)
    moreover have \( \llbracket \Phi' \rrbracket_{TESL} = \llbracket \Phi \sqcap \Phi' \rrbracket_{TESL} \)
        using TESL_interp_set_lifting decompose by blast
    moreover have \( \llbracket \Phi \sqcap \Phi' \rrbracket_{TESL} \supseteq \llbracket \Phi \rrbracket_{TESL} \cap \llbracket \Phi' \rrbracket_{TESL} \)
        by (simp add: TESL_interp_homo_append)
    moreover have \( \llbracket \Phi \rrbracket_{TESL} \supseteq \llbracket \Phi \rrbracket_{TESL} \cap \llbracket \Phi' \rrbracket_{TESL} \) by simp
    ultimately show \( \exists \Phi \) using TESL_interp_remdups_absorb by auto
qed simp

lemma TESL_interp_decreases_add_head:
    assumes \( \Phi \subseteq \Phi' \)
    shows \( \llbracket \varphi \# \Phi \rrbracket_{TESL} \supseteq \llbracket \varphi \# \Phi' \rrbracket_{TESL} \)
using assms TESL_interp_decreases_setinc by auto

lemma TESL_interp_decreases_add_tail:
    assumes \( \Phi \subseteq \Phi' \)
    shows \( \llbracket \Phi \# [\varphi] \rrbracket_{TESL} \supseteq \llbracket \Phi' \# [\varphi] \rrbracket_{TESL} \)
using TESL_interp_decreases_setinc[of assms] by (simp add: TESL_interp_homo_append dual_order.trans)

lemma TESL_interp_absorb1:
    assumes \( \Phi_1 \subseteq \Phi_2 \)
    shows \( \llbracket \Phi_1 \!= \# \Phi_2 \rrbracket_{TESL} = \llbracket \Phi_2 \rrbracket_{TESL} \)
by (simp add: Int_absorb1 TESL_interp_decreases_setinc TESL_interp_homo_append assms)

lemma TESL_interp_absorb2:
    assumes \( \Phi_2 \subseteq \Phi_1 \)
    shows \( \llbracket \Phi_1 \# \Phi_2 \rrbracket_{TESL} = \llbracket \Phi_1 \rrbracket_{TESL} \)

using TESL_interp_absorb1 TESL_interp_commute assms by blast

3.5 Some special cases

lemma NoSporadic_stable [simp]:
\( \langle [[ \Phi ]]_{TESL} \subseteq [[ \text{NoSporadic } \Phi ]]_{TESL} \rangle \)
proof -
  from filter_is_subset have \( \langle \text{set (NoSporadic } \Phi ) \subseteq \text{set } \Phi \rangle \).
  from TESL_interp_decreases_setinc[OF this] show \( ?\text{thesis} \).
  qed

lemma NoSporadic_idem [simp]:
\( \langle [[ \Phi ]]_{TESL} \cap [[ \text{NoSporadic } \Phi ]]_{TESL} = [[ \Phi ]]_{TESL} \rangle \)
using NoSporadic_stable by blast

lemma NoSporadic_setinc:
\( \langle \text{set (NoSporadic } \Phi ) \subseteq \text{set } \Phi \rangle \)
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 =
  TSchematic ⟨clock * instant_index⟩ (⟨τ_var⟩)

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) (⟨↓⟩ ⟨τ⟩ ⟨0⟩ ⟨↓⟩)
  — \( n \) \( \oplus \) \( δt \) \( ⇒ \) \( s \) 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⟩ (⟨↓⟩ ⟨τ⟩ ⟨0⟩ ⟨⊕⟩ ⇒ ⟨↓⟩)
  — \( c \) \( \oplus \) \( n \) 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 formulae.

```plaintext
fun counter_expr_eval :: \('\tau::linordered_field\) run \Rightarrow \text{cnt_expr} \Rightarrow \text{nat}
\langle \_, \_ \rangle\enterexpr
where
  \langle \_ \leftarrow _\cdot \_ \rangle \enterexpr = \text{run_tick_count_strictly } _\cdot \_ \text{ clk indx}
  \langle \_ \leftarrow _\cdot _\cdot \_ \rangle \enterexpr = \text{run_tick_count } _\cdot \_ \text{ clk indx}

fun symbolic_run_interpretation_primitive :: \('\tau::linordered_field\) constr \Rightarrow \text{'}\tau run set \langle \_ \rangle \text{prim}
where
  \langle \_ \uparrow \_ \rangle \text{prim} = (\_ . \text{hamlet } ((\text{Rep_run } _\cdot ) n \_ K))
  \langle \_ \odot n \_ \rangle \text{prim} = (\_ . \text{hamlet } ((\text{Rep_run } _\cdot ) n \_ K))
  \langle \_ \leftarrow _\cdot \_ \rangle \text{prim} = (\_ . \text{hamlet } ((\text{Rep_run } _\cdot ) i \_ K))
  \langle \_ \leftarrow _\cdot n \_ \rangle \text{prim} = (\_ . \text{hamlet } ((\text{Rep_run } _\cdot ) i \_ K))
  \langle \_ \leftarrow _\cdot \_ \rangle \text{prim} = (\_ . \text{hamlet } ((\text{Rep_run } _\cdot ) i \_ K))
  \langle \_ \leftarrow _\cdot _\cdot \_ \rangle \text{prim} = (\_ . \text{hamlet } ((\text{Rep_run } _\cdot ) i \_ K))
  \langle \_ \leftarrow _\cdot \_ \cdot \_ \rangle \text{prim} = (\_ . \text{hamlet } ((\text{Rep_run } _\cdot ) i \_ \cdot K))
```

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

fun symbolic_run_interpretation :: (τ::linordered_field) constr list ⇒ (τ::linordered_field) run set
where
⟨[[ ]]prim = {φ. True}⟩

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

definition consistent_context :: ((τ::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.

abbreviation initial_run :: ((τ::linordered_field) run) ((φ)) where
(φ) ≡ Abs_run ((λ_. (False, τcst 0)) ::nat ⇒ clock ⇒ (bool × τ tag_constr))

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

fun time_update :: (nat ⇒ clock ⇒ (τ::linordered_field) tag_constr ⇒ (nat ⇒ τ instant))
where
⟨time_update n K τ ϱ = (λn'. if K = K' ∧ n ≤ n' then (hamlet (ϱ n K), τ) else ϱ n' K')⟩

4.2 Rules and properties of consistence

lemma context_consistency_preservationI:
⟨consistent_context ((γ::(τ::linordered_field) constr)#Γ) ⇒ consistent_context Γ⟩
unfolding consistent_context_def by auto
— This is very restrictive

inductive context_independency :: ((τ::linordered_field) constr ⇒ τ constr list ⇒ bool) ((_ ⊢ _)⟩
where
NotTicks_independency:
⟨(K ⊢ n) /∈ set Γ ⇒ (K ⊬ n) ▼ Γ⟩
| Ticks_independency:
⟨(K ⊬ n) /∈ set Γ ⇒ (K ⊢ n) ▼ Γ⟩
| Timestamp_independency:
⟨(∃τ'. τ' = τ ∧ (K ⊬ n θ τ) ∈ set Γ) ⇒ (K ⊬ n θ 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.

\[ \bigcap (\lambda \gamma. [\gamma \text{prim}] \set \Gamma) = [\Gamma \text{prim}] \]

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

4.3.2 Expansion law

Similar to the expansion laws of lattices

\[ [\Gamma_1 \@ \Gamma_2 \text{prim}] = [\Gamma_1 \text{prim}] \cap [\Gamma_2 \text{prim}] \]

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

4.4 Equations for the interpretation of symbolic primitives

4.4.1 General laws

\[
\begin{align*}
\text{lemma } \text{symrun_interp_assoc:} & \quad [\Gamma_1 \@ (\Gamma_2 \@ \Gamma_3)]_{\text{prim}} = [\Gamma_1 \@ (\Gamma_2 \@ \Gamma_3)]_{\text{prim}} \\
& \quad \text{by auto} \\
\text{lemma } \text{symrun_interp_commute:} & \quad [\Gamma_1 \@ \Gamma_2]_{\text{prim}} = [\Gamma_2 \@ \Gamma_1]_{\text{prim}} \\
& \quad \text{by (simp add: symrun_interp_expansion inf_sup_aci(1))} \\
\text{lemma } \text{symrun_interp_left_commute:} & \quad [\Gamma_1 \@ (\Gamma_2 \@ \Gamma_3)]_{\text{prim}} = [\Gamma_2 \@ (\Gamma_1 \@ \Gamma_3)]_{\text{prim}} \\
& \quad \text{unfolding symrun_interp_expansion by auto} \\
\text{lemma } \text{symrun_interp_idem:} & \quad [\Gamma \@ \Gamma]_{\text{prim}} = [\Gamma]_{\text{prim}} \\
& \quad \text{using symrun_interp_expansion by auto} \\
\text{lemma } \text{symrun_interp_left_idem:} & \quad [\Gamma_1 \@ (\Gamma_2 \@ \Gamma_3)]_{\text{prim}} = [\Gamma_1 \@ \Gamma_2]_{\text{prim}} \\
& \quad \text{using symrun_interp_expansion by auto} \\
\text{lemma } \text{symrun_interp_right_idem:} & \quad [\Gamma_1 \@ (\Gamma_2 \@ \Gamma_3)]_{\text{prim}} = [\Gamma_1 \@ \Gamma_2]_{\text{prim}} \\
& \quad \text{unfolding symrun_interp_expansion by auto} \\
\text{lemmas } \text{symrun_interp_aci = symrun_interp_commute} & \quad \text{symrun_interp_assoc} \\
& \quad \text{symrun_interp_left_commute} \\
& \quad \text{symrun interp_left_idem} \\
\end{align*}
\]

— Identity element

\[
\begin{align*}
\text{lemma } \text{symrun_interp_neutral1:} & \quad [\emptyset \@ \Gamma]_{\text{prim}} = [\Gamma]_{\text{prim}} \\
& \quad \text{by simp} \\
\text{lemma } \text{symrun interp neutral2:} & \quad [\Gamma \@ \emptyset]_{\text{prim}} = [\Gamma]_{\text{prim}}
\end{align*}
\]
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.

lemma TESL_sem_decreases_head:
\[ \langle \Gamma \rangle_{\text{prim}} \supseteq \langle \gamma \# \Gamma \rangle_{\text{prim}} \]
by simp

lemma TESL_sem_decreases_tail:
\[ \langle \Gamma \rangle_{\text{prim}} \supseteq \langle \Gamma @ [\gamma] \rangle_{\text{prim}} \]
by (simp add: symrun_interp_expansion)

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

lemma symrun_interp_formula_stuttering:
assumes \( \gamma \in \text{set } \Gamma \) shows \( \langle \gamma \# \Gamma \rangle_{\text{prim}} = \langle \Gamma \rangle_{\text{prim}} \)
proof -
have \( \gamma \# \Gamma = [\gamma] @ \Gamma \) by simp
hence \( \langle \gamma \# \Gamma \rangle_{\text{prim}} = \langle [\gamma] \rangle_{\text{prim}} \cap \langle \Gamma \rangle_{\text{prim}} \)
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.

lemma symrun_interp_remdups_absorb:
\[ \langle \text{set } \Gamma \rangle_{\text{prim}} = \langle \text{remdups } \Gamma \rangle_{\text{prim}} \]
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.

lemma symrun_interp_set_lifting:
assumes \( \text{set } \Gamma = \text{set } \Gamma' \) shows \( \langle \Gamma \rangle_{\text{prim}} = \langle \Gamma' \rangle_{\text{prim}} \)
proof -
  have \( \text{set (remdups } \Gamma) = \text{set (remdups } \Gamma') \) by simp add: assms
  moreover have \( \text{fxpnt } \Gamma : \langle \lambda \gamma. \ [\gamma] \rangle_{\text{prim}} \cap \langle \text{set } \Gamma \rangle_{\text{prim}} \)
    by (simp add: symrun_interp_fixpoint)
  moreover have \( \text{fxpnt } \Gamma' : \langle \lambda \gamma. \ [\gamma] \rangle_{\text{prim}} \cap \langle \text{set } \Gamma' \rangle_{\text{prim}} \)
    by (simp add: symrun_interp_fixpoint)
  moreover have \( \text{set (remdups } \Gamma) \cap \langle \text{set } \Gamma \rangle_{\text{prim}} \)
    by (simp add: assms)
  ultimately show thesis using symrun_interp_remdups_absorb by auto
qed

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

theorem symrun_interp_decreases_setinc:
assumes \( \text{set } \Gamma \subseteq \text{set } \Gamma' \) shows \( \langle \gamma \# \Gamma \rangle_{\text{prim}} \supseteq \langle \Gamma \rangle_{\text{prim}} \)
proof -
obtain $\Gamma_r$ where decompose: $(\text{set } (\Gamma \ominus \Gamma_r) = \text{set } \Gamma')$ using assms by auto
hence $(\text{set } (\Gamma \ominus \Gamma_r) = \text{set } \Gamma')$ using assms by blast
moreover have $(\text{set } \Gamma) \cup (\text{set } \Gamma_r) = \text{set } \Gamma'$ using assms decompose by auto
moreover have $[[ \Gamma ]]_{prim} = [[[ \Gamma \ominus \Gamma_r ]]_{prim} \cap [[[ \Gamma_r ]]_{prim}}$
using symrun_interp_set_lifting decompose by blast
moreover have $[[ \Gamma ]]_{prim} \supseteq [[[ \Gamma_r ]]_{prim} \cap [[[ \Gamma_r ]]_{prim}}$
by (simp add: symrun_interp_expansion)
moreover have $[[ \Gamma ]]_{prim} = [[[ \Gamma_r ]]_{prim} \cap [[[ \Gamma_r ]]_{prim}}$
ultimately show ?thesis by simp
qed

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

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

lemma symrun_interp_absorb1:
  assumes $(\text{set } \Gamma_1 \subseteq \text{set } \Gamma_2)$
  shows $[[ \Gamma_1 \oplus \Gamma_2 ]]_{prim} = [[[ \Gamma_2 ]]_{prim}}$
by (simp add: Int_absorb1 symrun_interp_decreases_setinc symrun_interp_expansion assms)

lemma symrun_interp_absorb2:
  assumes $(\text{set } \Gamma_2 \subseteq \text{set } \Gamma_1)$
  shows $[[ \Gamma_1 \oplus \Gamma_2 ]]_{prim} = [[[ \Gamma_1 ]]_{prim}}$
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
\( \langle \Gamma, n \vdash \Psi \triangleright \Phi \equiv (\Gamma, n, \Psi, \Phi) \)\)

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.

**inductive operational_semantics_elim**

\[ (((\tau:\text{linordered_field}) \text{ config}) \Rightarrow \tau \text{ config} \Rightarrow \text{ bool}) \quad ((\_ \Rightarrow_e \_)) 70 \]

where

<table>
<thead>
<tr>
<th>sporadic_on_e1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A sporadic constraint can be ignored in the present and rejected into the future.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash ({K_1 \text{ sporadic } \tau \text{ on } K_2} # \Psi) \supset \Phi)) \Rightarrow_e ((\Gamma, n \vdash \Psi \supset (({K_1 \text{ sporadic } \tau \text{ on } K_2} # \Phi))) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sporadic_on_e2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash ({K_1 \text{ sporadic } \tau \text{ on } K_2} # \Psi) \supset \Phi)) \Rightarrow_e (({K_1 \uparrow n} # (K_2 \downarrow n &amp; \tau) # \Gamma), n \vdash \Psi \supset \Phi) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tagrel_e:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A relation between time scales has to be obeyed at every instant.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash ((\text{time-relation } [K_1, K_2] \in \mathbb{R}) # \Psi) \supset \Phi)) \Rightarrow_e (({\tau_{\text{var}}(K_1, n), \tau_{\text{var}}(K_2, n)} \in \mathbb{R} \Rightarrow \Gamma), n \vdash \Psi \Rightarrow ((\text{time-relation } [K_1, K_2] \in \mathbb{R}) # \Phi)) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>implies_e1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>An implication can be handled in the present by forbidding a tick of the master clock. The implication is copied back into the future because it holds for the whole run.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash (K_1 \text{ implies } K_2) # \Psi) \supset \Phi)) \Rightarrow_e ((K_1 \Rightarrow n) # (K_2 \Rightarrow n) # \Gamma), n \vdash \Psi \Rightarrow ((K_1 \text{ implies } K_2) # \Phi)) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>implies_e2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>It can also be handled in the present by making both the master and the slave clocks tick.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash (K_1 \text{ implies } K_2) # \Psi) \supset \Phi)) \Rightarrow_e ((K_1 \Rightarrow n) # (K_2 \Rightarrow n) # \Gamma), n \vdash \Psi \Rightarrow ((K_1 \text{ implies } K_2) # \Phi)) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>implies_not_e1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A negative implication can be handled in the present by forbidding a tick of the master clock. The implication is copied back into the future because it holds for the whole run.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash (K_1 \text{ implies not } K_2) # \Psi) \supset \Phi)) \Rightarrow_e ((K_1 \Rightarrow n) # \Gamma), n \vdash \Psi \Rightarrow ((K_1 \text{ implies not } K_2) # \Phi)) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>implies_not_e2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>It can also be handled in the present by making the master clock ticks and forbidding a tick on the slave clock.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash (K_1 \text{ implies not } K_2) # \Psi) \supset \Phi)) \Rightarrow_e ((K_1 \Rightarrow n) # (K_2 \Rightarrow n) # \Gamma), n \vdash \Psi \Rightarrow ((K_1 \text{ implies not } K_2) # \Phi)) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>timedelayed_e1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A timed delayed implication can be handled by forbidding a tick on the master clock.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash (K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) # \Psi) \supset \Phi)) \Rightarrow_e ((K_1 \Rightarrow n) # (K_2 \Rightarrow n) # \Gamma), n \vdash \Psi \Rightarrow ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) # \Phi)) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>timedelayed_e2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>It can also be handled by making the master clock tick and adding a constraint that makes the slave clock tick when the delay has elapsed on the measuring clock.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash (K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) # \Psi) \supset \Phi)) \Rightarrow_e ((K_1 \Rightarrow n) # (K_2 \Rightarrow n) # \Gamma), n \vdash \Psi \Rightarrow ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) # \Phi)) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>weakly_precedes_e:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A weak precedence relation has to hold at every instant.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash (K_1 \text{ weakly precedes } K_2) # \Psi) \supset \Phi)) \Rightarrow_e (((K_2 n, # K_1 n) \in (\lambda(x,y). x \leq y) \Rightarrow \Gamma), n \vdash \Psi \Rightarrow ((K_1 \text{ weakly precedes } K_2) # \Phi)) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>strictly_precedes_e:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A strict precedence relation has to hold at every instant.</td>
</tr>
<tr>
<td>[ ((\Gamma, n \vdash (K_1 \text{ strictly precedes } K_2) # \Psi) \supset \Phi)) \Rightarrow_e (((K_2 n, n &amp; K_1 n) \in \lambda(x,y). x \leq y) \Rightarrow \Gamma), n \vdash \Psi \Rightarrow ((K_1 \text{ strictly precedes } K_2) # \Phi)) ]</td>
</tr>
</tbody>
</table>
steps from an original configuration, then it can be reached in 
of the operational semantic step, its
| elims_part: 

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

\[
\begin{align*}
\text{intro_part:} & \quad (\Gamma_1, n_1 \vdash \Psi_1 \Rightarrow \Phi_1) \quad (\Gamma_2, n_2 \vdash \Psi_2 \Rightarrow \Phi_2) \\
\text{elims_part:} & \quad (\Gamma_1, n_1 \vdash \Psi_1 \Rightarrow \Phi_1) \quad (\Gamma_2, n_2 \vdash \Psi_2 \Rightarrow \Phi_2)
\end{align*}
\]

We introduce notations for the reflexive transitive closure of the operational semantic step, its transitive closure and its reflexive closure.

\[
\begin{align*}
\text{abbreviation operational_semantics_step_rtranclp} & \quad :\!\!: (\forall \tau:\!\!: \text{linordered_field}) \text{ config} \Rightarrow \tau \text{ config} \Rightarrow \text{bool} \\
\text{abbreviation operational_semantics_step_tranclp} & \quad :\!\!: (\forall \tau:\!\!: \text{linordered_field}) \text{ config} \Rightarrow \tau \text{ config} \Rightarrow \text{bool} \\
\text{abbreviation operational_semantics_step_reflclp} & \quad :\!\!: (\forall \tau:\!\!: \text{linordered_field}) \text{ config} \Rightarrow \tau \text{ config} \Rightarrow \text{bool} \\
\text{abbreviation operational_semantics_step_relpow} & \quad :\!\!: (\forall \tau:\!\!: \text{linordered_field}) \text{ config} \Rightarrow \text{nat} \Rightarrow \tau \text{ config} \Rightarrow \text{bool}
\end{align*}
\]

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:**
\[
\begin{align*}
\text{assumes } \langle C_1 \rightarrow^n C_2 \rangle \\
\text{assumes } \langle C_2 \rightarrow^* C_3 \rangle \\
\text{shows } \langle C_1 \rightarrow^* C_3 \rangle
\end{align*}
\]

**Using relcomp.relpowI of operational_semantics_step ^^ m**

by (simp add: operational_semantics_step.simps operational_semantics_elim.implies_not_e1 operational_semantics_elim.implies_e1)

**Lemma operational_semantics_elim.implies_not:**
\[
\begin{align*}
\text{by (simp add: operational_semantics_step.simps operational_semantics_elim.implies_not_e2)}
\end{align*}
\]

**Lemma operational_semantics_elim.implies:**
\[
\begin{align*}
\text{by (simp add: operational_semantics_step.simps operational_semantics_elim.implies_e2)}
\end{align*}
\]

**Abbreviation Cnext_solve:**
\[
\begin{align*}
\langle (\tau :: \text{linordered_field}) \text{ config } \Rightarrow \langle C_{\text{next solve}} \rangle \rangle
\end{align*}
\]

**Where Cnext Solve**
\[
\begin{align*}
\Rightarrow \langle \text{config set } \rangle (\langle C_{\text{next solve}} \rangle)
\end{align*}
\]

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

**Lemma Cnext_solveInstant:**
\[
\begin{align*}
\langle C_{\text{next solve}} \rangle (\Gamma, n \vdash [\emptyset \triangleright \Phi]) \supseteq \{ \Gamma, \text{Suc } n \vdash \Phi \triangleright [\emptyset] \}
\end{align*}
\]

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_sporadic:**
\[
\begin{align*}
\langle C_{\text{next solve}} \rangle (\Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi)) \\
\supseteq \{ \Gamma, n \vdash \Phi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi), \\
(K_1 \uparrow n) \# (K_2 \vdash n \# \tau) \# \Gamma), n \vdash \Phi \triangleright \Phi \}
\end{align*}
\]

by (simp add: operational_semantics_step.simps operational_semantics_elim.sporadic_on_el1 operational_semantics_elim.sporadic_on_el2)

**Lemma Cnext_solve_tagRel:**
\[
\begin{align*}
\langle C_{\text{next solve}} \rangle (\Gamma, n \vdash (\text{time-relation } [K_1, K_2] \subseteq R) \# \Psi) \triangleright \Phi)) \\
\supseteq \{ ((\tau \text{var}(K_1, n), \tau \text{var}(K_2, n)) \in R) \# \Gamma), n \vdash \Phi \triangleright ((\text{time-relation } [K_1, K_2] \subseteq R) \# \Phi) \}
\end{align*}
\]

by (simp add: operational_semantics_step.simps operational_semantics_elim.tagrel_e1)

**Lemma Cnext_solve_implies:**
\[
\begin{align*}
\langle C_{\text{next solve}} \rangle (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi)) \\
\supseteq \{ ((K_1 \leftarrow n) \# \Gamma), n \vdash \Phi \triangleright ((K_1 \text{ implies } K_2) \# \Phi), \\
(K_1 \uparrow n) \# (K_2 \vdash n \# \tau) \# \Gamma), n \vdash \Phi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \}
\end{align*}
\]

by (simp add: operational_semantics_step.simps operational_semantics_elim.implies_e1 operational_semantics_elim.implies_e2)

**Lemma Cnext_solve_implies_not:**
\[
\begin{align*}
\langle C_{\text{next solve}} \rangle (\Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi)) \\
\supseteq \{ ((K_1 \leftarrow n) \# \Gamma), n \vdash \Phi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi), \\
(K_1 \uparrow n) \# (K_2 \vdash n \# \tau) \# \Gamma), n \vdash \Phi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \}
\end{align*}
\]

by (simp add: operational_semantics_step.simps operational_semantics_elim.implies_not_e1 operational_semantics_elim.implies_not_e2)

**Lemma Cnext_solve_timeDelayed:**
\[
\begin{align*}
\langle C_{\text{next solve}} \rangle (\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi)) \\
\supseteq \{ ((K_1 \leftarrow n) \# \Gamma), n \vdash \Phi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi), \\
(K_1 \uparrow n) \# (K_2 \vdash n \# \tau) \# \Gamma), n \vdash \Phi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \}
\end{align*}
\]
5.2. BASIC LEMMAS

\[(K_1 \uparrow n) \# (K_2 \ominus n \oplus \delta \tau \Rightarrow K_3) \# \Gamma), n
\vdash \psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \}\]
by (simp add: operational_semantics_step.simps
operational_semantics_elim.timedelayed_e1
operational_semantics_elim.timedelayed_e2)

lemma Cnext_solve_weakly_precedes:
\[\langle C_{next}(\Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi) \rangle
\supseteq \{ ((\llbracket \# \leq K_2 n, \llbracket \# K_1 n \rrbracket \in (\lambda(x,y). x \leq y) \# \Gamma), n
\vdash \psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \}\]
by (simp add: operational_semantics_step.simps
operational_semantics_elim.weakly_precedes_e)

lemma Cnext_solve_strictly_precedes:
\[\langle C_{next}(\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi) \rangle
\supseteq \{ ((\llbracket \# \leq K_2 n, \llbracket \# K_1 n \rrbracket \in (\lambda(x,y). x \leq y) \# \Gamma), n
\vdash \psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi) \}\]
by (simp add: operational_semantics_step.simps
operational_semantics_elim.strictly_precedes_e)

lemma Cnext_solve_kills:
\[\langle C_{next}(\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi) \rangle
\supseteq \{ ((K_1 \downarrow n) \# \# \Gamma), n \vdash \psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi),
((K_1 \uparrow n) \# (K_2 \neg \downarrow n) \# \Gamma), n \vdash \psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \}\]
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:
\[\langle [], 0 \vdash [] \triangleright [] \rangle \Rightarrow^* \langle [], k \vdash [] \triangleright [] \rangle\]
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 ⇒ 'τ
where
  ⟨ | [ TESL ≥ i = \{ \varrho. \exists n ≥ i. hamlet ((Rep_run \varrho) n K1) ∧ time ((Rep_run \varrho) n K2) = τ} ]⟩ | [ TESL ≥ i = \{ \varrho. \forall n ≥ i. R (time ((Rep_run \varrho) n K1), time ((Rep_run \varrho) n K2))} ]⟩ | [ TESL ≥ i = \{ \varrho. \forall n ≥ i. hamlet ((Rep_run \varrho) n master) → hamlet ((Rep_run \varrho) n slave)} ]⟩ | [ TESL ≥ i = \{ \varrho. \forall n ≥ i. hamlet ((Rep_run \varrho) n master) → ¬ hamlet ((Rep_run \varrho) n slave)} ]⟩ | [ TESL ≥ i = \{ \varrho. \forall n ≥ i. (run_tick_count \varrho K2 n) ≤ (run_tick_count \varrho K1 n)\} ]⟩
The denotational interpretation of TESL formulae can be unfolded into the stepwise interpretation.

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

lemma TESL_interp_unfold_stepwise_tagrelgen:
  \[ [ \text{time-relation } \lfloor K_1, K_2 \rfloor \in R ]_{TESL} = \bigcap \{ Y. \exists n::\text{nat}. Y = [ \text{time-relation } \lfloor K_1, K_2 \rfloor \in R ]_{TESL} \geq n \} \]
by auto

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

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

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

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

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

lemma TESL_interp_unfold_stepwise_kills:
  \[ [ \text{master kills slave } ]_{TESL} = \bigcap \{ Y. \exists n::\text{nat}. Y = [ \text{master kills slave } ]_{TESL} \geq n \} \]
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 \([ \varphi::\tau::\text{linordered_field TESL_atomic} ]_{TESL} \]
  shows \([ \varphi ]_{TESL} = \bigcup \{ Y. \exists n::\text{nat}. Y = [ \varphi ]_{TESL} \geq n \} \]
proof -
  from positive_atom.elims(2)[OF assms] obtain u v w where \( \varphi = (u \text{ sporadic } v \text{ on } w) \) by blast
  with TESL_interp_unfold_stepwise_sporadicon show \?thesis by simp
Negative atomic formulae are unfolded as the intersection of the stepwise interpretations.

\[
\text{theorem TESL_interp_unfold_stepwise_negative_atoms:}
\begin{align*}
\text{assumes} & \langle \neg \text{positive_atom} \, \varphi \rangle \\
\text{shows} & \langle [\varphi]_{\text{TESL}} = \bigcap \{Y. \exists n::\text{nat}. Y = [\varphi]_{\text{TESL}} \geq n\} \rangle \\
\end{align*}
\]
proof (cases \varphi)
  case SporadicOn thus ?thesis using assms by simp
  next
  case TagRelation thus ?thesis using TESL_interp_unfold_stepwise_tagrelgen by simp
  next
  case Implies thus ?thesis using TESL_interp_unfold_stepwise_implies by simp
  next
  case ImpliesNot thus ?thesis using TESL_interp_unfold_stepwise_implies_not by simp
  next
  case TimeDelayedBy thus ?thesis using TESL_interp_unfold_stepwise_timedelayed by simp
  next
  case WeaklyPrecedes thus ?thesis using TESL_interp_unfold_stepwise_weakly_precedes by simp
  next
  case StrictlyPrecedes thus ?thesis using TESL_interp_unfold_stepwise_strictly_precedes by simp
  next
  case Kills thus ?thesis using TESL_interp_unfold_stepwise_kills by simp
qed

Some useful lemmas for reasoning on properties of sequences.

\[\text{lemma forall_nat_expansion:} \quad \langle \forall n \geq (n_0::\text{nat}). P n \rangle = (P n_0 \land (\forall n \geq \text{Suc } n_0. P n))\]
proof =
  have \langle \forall n \geq (n_0::\text{nat}). P n \rangle = (\forall n. (n = n_0 \lor n > n_0) \longrightarrow 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

\[\text{lemma exists_nat_expansion:} \quad \langle \exists n \geq (n_0::\text{nat}). P n \rangle = (P n_0 \lor (\exists n \geq \text{Suc } n_0. P n))\]
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

\[\text{lemma forall_nat_set_suc:} \quad \langle \forall m \geq n. P \times m \rangle = (x. P x n) \cap \{x. \forall m \geq \text{Suc } n. P x m\}\]
proof
  { fix x assume h:x \in (x. \forall m \geq n. P x m)\)
    hence (P x n) by simp
    moreover from h have \langle x \in (x. \forall m \geq \text{Suc } n. P x m)\rangle by simp
    ultimately have \langle x \in (x. P x n) \cap \{x. \forall m \geq \text{Suc } n. P x m\}\rangle by simp
  } thus (x. \forall m \geq n. P x m) \subseteq (x. P x n) \cap \{x. \forall m \geq \text{Suc } n. P x m\} ..
next
{ fix x assume h: \( x \in \{ x \cdot P \cdot x \} \cap \{ x \cdot \forall m \geq \text{Suc} \cdot n \cdot P \cdot x \mbox{m} \}\)
  hence \( P \cdot x \cdot n \) by simp
  moreover from h have \( \forall n \geq \text{Suc} \cdot n \cdot P \cdot x \cdot n \) by simp
  ultimately have \( \forall n \geq \text{Suc} \cdot n \cdot P \cdot x \cdot n \) using forall_nat_expansion by blast
} thus \( \{ x \cdot P \cdot x \} \cap \{ x \cdot \forall m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot n \} \subseteq \{ x \cdot \forall m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot n \} \) ..
qed

lemma exists_nat_set_suc: \( \{ x \cdot \exists m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot m \} = \{ x \cdot P \cdot x \} \cup \{ x \cdot \exists m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot m \} \)
proof
{ fix x assume h: \( x \in \{ x \cdot \exists m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot m \} \)
  hence \( x \in \{ x \cdot \exists m \cdot (m = n \lor m \geq \text{Suc} \cdot n) \land P \cdot x \cdot m \} \)
  using Suc_le_eq antisym_conv2 by fastforce
  hence \( x \in \{ x \cdot P \cdot x \} \cup \{ x \cdot \exists m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot n \} \) by blast
} thus \( \{ x \cdot \exists m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot m \} \subseteq \{ x \cdot P \cdot x \} \cup \{ x \cdot \exists m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot n \} \) ..
next
{ fix x assume h: \( x \in \{ x \cdot P \cdot x \} \cup \{ x \cdot \exists m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot m \} \)
  hence \( x \in \{ x \cdot \exists m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot m \} \) using Suc_le0 by blast
} thus \( \{ x \cdot P \cdot x \} \cup \{ x \cdot \exists m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot m \} \subseteq \{ x \cdot \exists m \geq \text{Suc} \cdot n \cdot P \cdot x \cdot n \} \) ..
qed

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.

lemma TESL_interp_stepwise_sporadicon_coind_unfold:
\[
\begin{align*}
& \{ K_1 \text{ sporadic } \tau \text{ on } K_2 \}^{TESL \geq n} = \\
& \{ K_1 \uparrow n \}_{prim} \cap \{ K_2 \downarrow n \circ \tau \}_{prim} \quad \text{— sporadicon_e2} \\
& \cup \{ K_1 \text{ sporadic } \tau \text{ on } K_2 \}^{TESL \geq \text{Suc} \cdot n} \quad \text{— sporadicon_e1}
\end{align*}
\]
unfolding TESL_interpretation_atomic_stepwise.simps(1)
symbolic_run_interpretation_primitive.simps(1,6)
using exists_nat_set_suc[of \( \langle n \rangle \cdot \lambda \theta \cdot \text{hamlet} \ (\text{Rep}\_run \ \theta \ n \ K_1) \)
\( \land \text{time} \ (\text{Rep}\_run \ \theta \ n \ K_2) = \tau \rangle \]
by (simp add: Collect_conj_eq)

lemma TESL_interp_stepwise_tagrel_coind_unfold:
\[
\begin{align*}
& \{ \text{time}\text{-relation} [K_1, K_2] \in R \}^{TESL \geq n} = \\
& \{ \tau_{var} (K_1, n), \tau_{var} (K_2, n) \} \in R \}_{\text{prim}} \\
& \cup \{ \text{time}\text{-relation} [K_1, K_2] \in R \}^{TESL \geq \text{Suc} \cdot n} \quad \text{— tagrel_e}
\end{align*}
\]
proof
{ have \( \langle \theta, \forall n \geq n \cdot R \ (\text{time} \ (\text{Rep}\_run \ \theta \ n \ K_1), \text{time} \ (\text{Rep}\_run \ \theta \ n \ K_2)) \rangle = \langle \theta, R \ (\text{time} \ (\text{Rep}\_run \ \theta \ n \ K_1), \text{time} \ (\text{Rep}\_run \ \theta \ n \ K_2)) \rangle \)
  \( \land \langle \theta, \forall n \geq \text{Suc} \cdot n \cdot R \ (\text{time} \ (\text{Rep}\_run \ \theta \ n \ K_1), \text{time} \ (\text{Rep}\_run \ \theta \ n \ K_2)) \rangle \)
  using forall_nat_set_suc[of \( \langle n \rangle \cdot \lambda x y \cdot R \ (\text{time} \ (\text{Rep}\_run x \ y \ K_1), \text{time} \ (\text{Rep}\_run x \ y \ K_2)) \rangle \]
  by simp
} thus ?thesis by auto
qed

lemma TESL_interp_stepwise_implies_coind_unfold:
\[
\begin{align*}
& \{ \text{master implies slave} \}^{TESL \geq n} = \\
& \{ \text{master } \uparrow n \}_{prim} \quad \text{— implies_e1} \\
& \cup \{ \text{master } \uparrow n \}_{prim} \cap \{ \text{slave } \uparrow n \}_{prim} \quad \text{— implies_e2} \\
& \cap \{ \text{master implies slave} \}^{TESL \geq \text{Suc} \cdot n} \quad \text{— implies_e2}
\end{align*}
\]
6.2. COINDUCTION UNFOLDING PROPERTIES

proof

have \((\varphi \land \forall n \geq n. \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave}))\)

\(= (\varphi \land \forall n \geq n. \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave}))\)

\(\land (\varphi \land \forall n \geq n. \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave}))\)

using \(\forall n \geq n. \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave})\) by simp

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

qed

lemma TESL_interp_stepwise_implies_not_coind_unfold:

\(\{ \text{master implies not slave} \}^{TESL}_{\leq n} = \)

\(\{ \text{master} \rightarrow n \}_{prim} \cap \{ \text{slave} \rightarrow n \}_{prim} \)

\(\cup \{ \text{master} \rightarrow n \}_{prim} \cap \{ \text{slave} \rightarrow n \}_{prim} \)

\(\cap \{ \text{master} \rightarrow n \}_{prim} \cap \{ \text{slave} \rightarrow n \}_{prim} \)

proof

have \((\varphi \land \forall n \geq n. \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave}))\)

\(= (\varphi \land \forall n \geq n. \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave}))\)

\(\land (\varphi \land \forall n \geq n. \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave}))\)

using \(\forall n \geq n. \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave})\) by simp

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

qed

lemma TESL_interp_stepwise_timedelayed_coind_unfold:

\(\{ \text{master time-delayed by } \delta \tau \text{ on measuring implies slave} \}^{TESL}_{\geq n} = \)

\(\{ \text{master} \rightarrow n \}_{prim} \cap \{ \text{slave} \rightarrow n \}_{prim} \)

\(\cup \{ \text{master} \rightarrow n \}_{prim} \cap \{ \text{slave} \rightarrow n \}_{prim} \)

\(\cap \{ \text{master} \rightarrow n \}_{prim} \cap \{ \text{slave} \rightarrow n \}_{prim} \)

proof

let \(\text{?prop} = (\lambda \varphi \circ n. \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave}))\)

\(\text{(let measured_time} = \text{time} ((\text{Rep}_\varphi \circ n) \circ \text{measuring}) \text{ in}\)

\(\forall p \geq n. \text{first_time} \varphi \circ \text{measuring} \circ p \circ (\text{measured_time} + \delta \tau)\)

\(\rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave}))\)

have \(\{ \varphi. \forall n \geq n. \text{?prop} \circ n \}\land \{ \varphi. \forall n \geq n. \text{?prop} \circ \circ n \}\)

using \(\forall n \geq n. \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{master}) \rightarrow \text{hamlet} ((\text{Rep}_\varphi \circ n) \circ \text{slave})\) by blast

also have \(\ldots = \{ \varphi \text{?prop} \circ n \}\)

\(\land \{ \text{master time-delayed by } \delta \tau \text{ on measuring implies slave} \}^{TESL}_{\geq n} \)

by simp

finally show \(\text{thesis by auto}\)

qed

lemma TESL_interp_stepwise_weakly_precedes_coind_unfold:

\(\{ \text{weakly precedes } K_1 \}^{TESL}_{\leq n} = \)

\(\{ \text{weakly precedes } K_2 \}^{TESL}_{\leq n} \cap \{ \text{weakly precedes } K_2 \}^{TESL}_{\leq n} \)

proof

have \((\varphi. \forall p \geq n. \text{run_tick_count} \circ \text{K}_2 \circ p \leq (\text{run_tick_count} \circ K_1 \circ p)\)

\(= (\varphi. \text{run_tick_count} \circ K_2 \circ p \leq (\text{run_tick_count} \circ K_1 \circ p))\)

\(\land (\varphi. \forall p \geq n. \text{run_tick_count} \circ K_2 \circ p \leq (\text{run_tick_count} \circ K_1 \circ p))\)

using \(\forall n \geq n. \text{run_tick_count} \circ K_2 \circ n \leq (\text{run_tick_count} \circ K_1 \circ n)\)

by simp

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

qed
lemma TESL_interp_stepwise_strictly_precedes_coind_unfold:
\[
\begin{align*}
[ K_1 \text{ strictly precedes } K_2 ]_{\text{TESL}_n} &= \quad \text{— rule strictly_precedes_e} \\
\text{let } \phi \in \{ \text{run_tick_count } \varnothing K_2 \} &\Rightarrow \text{— rule } \text{writes_e1} \\
\text{let } ?\text{ticks} &\in \{ ?\text{ticks } n K_1 \} \\
\text{thus } ?\text{kills } n \varnothing &\text{ by simp} \\
\text{moreover have } (\neg ?\text{ticks } n \varnothing K_1 \land ?\text{dead } n \varnothing K_2) &\lor (\neg ?\text{kills } (\text{Suc } n) \varnothing) \\
\text{using } \text{Suc_leD} &\text{ by blast} \\
\text{thus } ?\text{kills } n \varnothing &\text{ by blast} \\
\text{next} \\
\text{let } \phi \in \{ ?\text{ticks } n \varnothing K_1 \} \\
\text{thus } (\neg ?\text{ticks } n \varnothing K_1 \lor ?\text{kills } (\text{Suc } n) \varnothing) &\lor (\neg ?\text{ticks } n \varnothing K_2) \\
\text{using } \text{Suc_leD} &\text{ by blast} \\
\text{thus } ?\text{kills } n \varnothing &\text{ by blast} \\
\text{qed}
\end{align*}
\]

proof

have \( \langle \varnothing, \forall p \geq n. \) (run_tick_count \varnothing K_2 p) \leq (run_tick_count_strictly \varnothing K_1 p) \)
\( \cap \{ \varnothing, \forall p \geq \text{Suc } n. \) (run_tick_count \varnothing K_2 p) \leq (run_tick_count_strictly \varnothing K_1 p) \)
us

using forall_nat_setSuc[of n] \( \langle \lambda \varnothing n. \) (run_tick_count \varnothing K_2 n) \)
\( \leq (\text{run_tick_count_strictly } \varnothing K_1 n) \)
by simp
thus \?thesis by auto
qed
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
  "[ [ ] ] TESL ≥ n = {ϱ. True}

lemma TESL_interpretation_stepwise_fixpoint:
  by (induction Φ, simp, auto)

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

lemma TESL_interpretation_stepwise_zero:
  by (induction ϕ, simp+)

lemma TESL_interpretation_stepwise_zero':
  by (induction Φ, simp, simp add: TESL_interpretation_stepwise_zero)

lemma TESL_interpretation_stepwise_cons_morph:
  by auto

theorem TESL_interp_stepwise_composition:
  by (induction Φ1, 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.
by (simp add: TESL_interp_stepwise_composition
symrun_interp_expansion inf_assoc inf_left_commute)

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:
  \[ \Gamma, n \vdash \Phi \rightarrow \Box \Rightarrow \Gamma, \Suc n \vdash \Phi \rightarrow \Box \Rightarrow \]
proof -
  have \[ \Gamma, n \vdash \Box \rightarrow \Box \Rightarrow \Gamma \rightarrow \Box \Rightarrow \Gamma, \Suc n \vdash \Box \rightarrow \Box \Rightarrow \]
  by simp
  moreover have \[ \Gamma, \Suc n \vdash \Box \rightarrow \Box \Rightarrow \Gamma \rightarrow \Box \Rightarrow \Gamma, \Suc n \vdash \Box \rightarrow \Box \Rightarrow \]
  by simp
  ultimately show \?thesis 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_sporadicon_cases:
  \[ \Gamma, n \vdash (K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi \rightarrow \Box \Rightarrow \Gamma \rightarrow \Box \Rightarrow \Gamma \]
proof -
  have \[ \Gamma, n \vdash \Box \rightarrow \Box \Rightarrow (K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi \rightarrow \Box \Rightarrow \]
  by simp
  moreover have \[ \Gamma \rightarrow \Box \Rightarrow (K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi \rightarrow \Box \Rightarrow \]
  by simp
  ultimately show \?thesis

qed

lemma HeronConf_interp_stepwise_tagrel_cases:
  \[ \Gamma, n \vdash ((\text{time-relation } [K_1, K_2]) \in R) \# \Psi \rightarrow \Box \Rightarrow \Gamma \rightarrow \Box \Rightarrow \Gamma \]
proof -
6.3. INTERPRETATION OF CONFIGURATIONS

have \((\Gamma, n) \vdash (\text{time-relation} \ [K_1, K_2] \in R) \# \Psi \triangleright \Phi\)_{\text{config}}
\[\begin{aligned}
&\Gamma_{\text{prim}} \land (\text{time-relation} \ [K_1, K_2] \in R) \# \Psi \triangleright \Phi_{\text{TESL}} \geq n \land (\Phi)_{\text{TESL}} \geq \text{Suc} \ n, \text{by simp}
\end{aligned}\]
moreover have \((\langle \tau_{\text{var}}(K_1, n), \tau_{\text{var}}(K_2, n) \rangle \in R) \# \Gamma\), \(n \vdash \Psi \triangleright (\text{time-relation} \ [K_1, K_2] \in R) \# \Phi_{\text{config}}
\[\begin{aligned}
&\Gamma_{\text{prim}} \land (\text{time-relation} \ [K_1, K_2] \in R) \# \Psi_{\text{TESL}} \geq n \\
&\land (\Phi)_{\text{TESL}} \geq \text{Suc} \ n, \text{by simp}
\end{aligned}\]
ultimately show \(?thesis
thefor
qed

lemma HeronConf_intermpstepwise_implies_cases:
\[\begin{aligned}
&\langle \Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \triangleright \Phi\rangle_{\text{config}}
&\Gamma_{\text{prim}} \land (K_1 \text{ implies } K_2) \# \Psi \triangleright \Phi_{\text{TESL}} \geq n \land (\Phi)_{\text{TESL}} \geq \text{Suc} \ n, \text{by simp}
\end{aligned}\]
moreover have \((\langle K_1 \vdash n \# \Gamma \rangle, n \vdash \Psi \triangleright (K_1 \text{ implies } K_2) \# \Phi\rangle_{\text{config}}
\[\begin{aligned}
&\Gamma_{\text{prim}} \land (K_1 \text{ implies } K_2) \# \Psi_{\text{TESL}} \geq n \\
&\land (\Phi)_{\text{TESL}} \geq \text{Suc} \ n, \text{by simp}
\end{aligned}\]
moreover have \((\langle K_1 \vdash n \# \Gamma \rangle, n \vdash \Psi \triangleright (K_1 \text{ implies } K_2) \# \Phi\rangle_{\text{config}}
\[\begin{aligned}
&\Gamma_{\text{prim}} \land (K_1 \text{ implies } K_2) \# \Psi_{\text{TESL}} \geq n \\
&\land (\Phi)_{\text{TESL}} \geq \text{Suc} \ n, \text{by simp}
\end{aligned}\]
ultimately show \(?thesis
thefor
have f1: \((\langle K_1 \vdash n \# \Gamma \rangle, n \vdash (K_1 \text{ implies } K_2) \# \Psi \triangleright \Phi\rangle_{\text{config}}
\[\begin{aligned}
&(\langle K_1 \vdash n \# \Gamma \rangle, n \vdash (K_1 \text{ implies } K_2) \# \Psi_{\text{TESL}} \geq n \\
&\land (\Phi)_{\text{TESL}} \geq \text{Suc} \ n, \text{by simp}
\end{aligned}\]
using TESL_intermpstepwise_implies_coind_unfold
TESL_interpretation_stepwise_cons_morph by blast
have \((\langle K_1 \vdash n \# \Gamma \rangle, n \vdash (K_1 \text{ implies } K_2) \# \Psi \triangleright \Phi\rangle_{\text{config}}
\[\begin{aligned}
&(\langle K_1 \vdash n \# \Gamma \rangle, n \vdash (K_1 \text{ implies } K_2) \# \Psi_{\text{TESL}} \geq n \\
&\land (\Phi)_{\text{TESL}} \geq \text{Suc} \ n, \text{by simp}
\end{aligned}\]
using f1 by (simp add: int_left_commute inf_assoc)
thefor thus \(?thesis by (simp add: Int_Un_distrib2 inf_assoc)
qed

lemma HeronConf_intermpstepwise_implies_not_cases:
\[\begin{aligned}
&\langle \Gamma, n \vdash (K_1 \text{ implies not } K_2) \# \Psi \triangleright \Phi\rangle_{\text{config}}
&\Gamma_{\text{prim}} \land (K_1 \text{ implies not } K_2) \# \Psi \triangleright \Phi_{\text{TESL}} \geq n \land (\Phi)_{\text{TESL}} \geq \text{Suc} \ n, \text{by simp}
\end{aligned}\]
by simp

moreover have \( \{ (K_1 \supset n) \land \Gamma \} \), \( \n \vdash \Psi \vdash (\langle (K_1 \implies \neg n) \land \Gamma \rangle)_{\text{config}} \)

\( \cap \{ (K_1 \supset n) \land \Gamma \} )_{\text{prim}} \cap \{ (\Psi)_{\text{TESL}} \land \n \}

\( \cap \{ (K_1 \supset n) \land \Gamma \})_{\text{prim}} \cap \{ (\Psi)_{\text{TESL}} \land \n \}

by simp

moreover have \( \{ (K_1 \supset n) \land (K_2 \supset n) \land \Gamma \} \), \( \n \vdash \Psi \vdash (\langle (K_1 \implies \neg n) \land \Gamma \rangle)_{\text{config}} \)

\( \cap \{ (K_1 \supset n) \land (K_2 \supset n) \land \Gamma \})_{\text{prim}} \cap \{ (\Psi)_{\text{TESL}} \land \n \}

\( \cap \{ (K_1 \supset n) \land (K_2 \supset n) \land \Gamma \})_{\text{prim}} \cap \{ (\Psi)_{\text{TESL}} \land \n \}

by simp

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

proof -

have f1: \( \langle (K_1 \supset n)_{\text{prim}} \cup (K_1 \supset n)_{\text{prim}} \rangle \cap (K_1 \implies \neg n) \rangle_{\text{prim}} \cap (\{ (\Psi)_{\text{TESL}} \land \n \}

by force

then have \( \{ (\Gamma)_{\text{prim}} \cup (\{ K_1 \supset n \}_{\text{prim}} \cap (K_2 \supset n) \land \Gamma)_{\text{prim}} \cap (\{ (\Psi)_{\text{TESL}} \land \n \}

\cap (\{ K_1 \supset n \}_{\text{prim}} \cap (\{ (\Psi)_{\text{TESL}} \land \n \}

by simp using \text{TESL_interpretation_stepwise_cons_morph} by blast

have \( \{ (\Gamma)_{\text{prim}} \cup (\{ K_1 \supset n \}_{\text{prim}} \cap (K_2 \supset n) \land \Gamma)_{\text{prim}} \cap (\{ (\Psi)_{\text{TESL}} \land \n \}

\cap (\{ K_1 \supset n \}_{\text{prim}} \cap (\{ (\Psi)_{\text{TESL}} \land \n \}

by simp using f1 by (simp add: inf_left_commute inf_assoc)

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

qed

lemma HeronConf_interp_stepwise_timedelayed_cases:

\( \langle (\Gamma)_{\text{prim}} \cap (\{ (K_1 \supset n \}_{\text{prim}} \cap (K_2 \supset n) \land \Gamma) \cap (\{ (\Psi)_{\text{TESL}} \land \n \}

\cap (\{ K_1 \supset n \}_{\text{prim}} \cap (\{ (\Psi)_{\text{TESL}} \land \n \)

by simp

moreover have \( \{ (K_1 \supset n) \land \Gamma \} , \n \vdash \Psi \vdash (\langle (K_1 \supset n) \land \Gamma \rangle)_{\text{config}} \)

\( \cap \{ (K_1 \supset n) \land \Gamma \})_{\text{prim}} \cap \{ (\Psi)_{\text{TESL}} \land \n \}

by simp

moreover have \( \{ (K_1 \supset n) \land (K_2 \supset n) \land \Gamma \} , \n \vdash \Psi \vdash (\langle (K_1 \supset n) \land (K_2 \supset n) \land \Gamma \rangle)_{\text{config}} \)

\( \cap \{ (K_1 \supset n) \land (K_2 \supset n) \land \Gamma \})_{\text{prim}} \cap \{ (\Psi)_{\text{TESL}} \land \n \}

by simp

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

proof -

have \( \{ (\Gamma)_{\text{prim}} \cap (\{ (K_1 \supset n \}_{\text{prim}} \cap (K_2 \supset n) \land \Gamma) \cap (\{ (\Psi)_{\text{TESL}} \land \n \)

by simp using \text{TESL_interpretation_stepwise_cons_morph}

proof -
6.3. INTERPRETATION OF CONFIGURATIONS

lemma HeronConf_interp_stepwise_strictly_precedes_cases:
\[ \begin{align*}
\text{have } & \left( \begin{array}{l}
( K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \ # \ \Psi \\
( K_1 \rightleftharpoons n \ #\_\_ \# \_\_ K_3) \ # \ \phi \end{array} \right)_{T\text{ESL}} \geq n \\
\text{proof -} \end{align*} \]

then show ?thesis
by (simp add: Int_assoc Int_left_commute)
qed

lemma HeronConf_interp_stepwise_weakly_precedes_cases:
\[ \begin{align*}
\text{have } & \left( \begin{array}{l}
( K_1 \text{ weakly precedes } K_2) \ # \ \Psi \ \Diamond \ \phi \end{array} \right)_{config} \\
\text{proof -} \end{align*} \]

then show ?thesis by (simp add: inf_assoc inf_sup_distrib2)
qed

lemma HeronConf_interp_stepwise_strictly_precedes_cases:
\[ \begin{align*}
\text{have } & \left( \begin{array}{l}
( K_1 \text{ strictly precedes } K_2) \ # \ \Psi \ \Diamond \ \phi \end{array} \right)_{config} \\
\text{proof -} \end{align*} \]

then show ?thesis by (simp add: Int_assoc Int_left_commute)
qed

lemma HeronConf_interp_stepwise_weakly_precedes_cases:
\[ \begin{align*}
\text{have } & \left( \begin{array}{l}
( K_1 \text{ weakly precedes } K_2) \ # \ \Psi \ \Diamond \ \phi \end{array} \right)_{config} \\
\text{proof -} \end{align*} \]

then show ?thesis by (simp add: inf_assoc inf_sup_distrib2)
qed
lemma HeronConf_interp_stepwise_kills_cases:

\[
\{ \Gamma, n \vdash (K_1 \text{ kills } K_2) \# \Psi \} \backslash \Phi \}_{\text{config}}
\]

\[
\bigcup \{ \Gamma \vdash (K_1 \vdash \neg \uparrow \, n) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \}_{\text{config}}
\]

\[
\bigcup \{ \Gamma \vdash (K_1 \uparrow \, n) \# (K_2 \vdash \neg \uparrow \, \geq \, n) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \}_{\text{config}}
\]

proof -

have \[
\{ \Gamma, n \vdash (K_1 \text{ kills } K_2) \# \Psi \} \not\triangleright \Phi \}_{\text{config}}
\]

\[
\{ \Gamma \}, n \vdash (K_1 \text{ kills } K_2) \# \Psi \} \not\triangleright \Phi \}_{\text{config}}
\]

by simp

moreover have \[
\{ \Gamma \vdash (K_1 \vdash \neg \uparrow \, n) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \}_{\text{config}}
\]

\[
\{ \Gamma \vdash (K_1 \vdash \neg \uparrow \, n) \# \Gamma \}_{\text{prim}} \cap \{ \Psi \}_{\text{TESL} \geq \, n}
\]

by simp

moreover have \[
\{ \Gamma \vdash (K_1 \uparrow \, n) \# (K_2 \vdash \neg \uparrow \, \geq \, n) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \}_{\text{config}}
\]

\[
\{ \Gamma \vdash (K_1 \uparrow \, n) \# (K_2 \vdash \neg \uparrow \, \geq \, n) \# \Gamma \}_{\text{prim}} \cap \{ \Psi \}_{\text{TESL} \geq \, n}
\]

ultimately show ?thesis

proof -

have \[
\{ \Gamma \vdash (K_1 \text{ kills } K_2) \# \Psi \}_{\text{TESL} \geq \, n}
\]

\[
\{ \Gamma \vdash (K_1 \vdash \neg \uparrow \, n) \}_{\text{prim}} \cup \{ \Gamma \vdash (K_1 \vdash \neg \uparrow \, n) \}_{\text{prim}} \cap \{ \Psi \}_{\text{TESL} \geq \, n}
\]

using TESL_interp_stepwise_coind_unfold

TESL_interpretation_stepwise_coind_morph by blast

thus ?thesis by auto

qed
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.

\[
\text{theorem solve_start:} \quad \text{shows } \langle [\llbracket \Psi \rrbracket_{TESL} = [\llbracket \Psi \rrbracket_{TESL} \geq 0] \rangle \quad \text{proof -} \\
\text{have } \langle [\llbracket \Psi \rrbracket_{TESL} = [\llbracket \Psi \rrbracket_{TESL} \geq 0] \rangle \\
\text{by (simp add: TESL_interpretation_stepwise_zero')} \\
\text{moreover have } [\llbracket \Psi \rrbracket_{TESL} \geq 0]_{config} = \langle [\llbracket \Psi \rrbracket_{TESL} \geq 0] \rangle_{config} \\
\text{by simp} \\
\text{ultimately show ?thesis by auto} \quad \text{qed}
\]

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.

\[
\text{lemma sound_reduction:} \quad \text{assumes } (\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle \leftrightarrow (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)) \\
\text{shows } \langle [\llbracket \Gamma_1 \rrbracket_{prim} \cap [\llbracket \Psi_1 \rrbracket_{TESL} \geq n_1] \cap [\llbracket \Phi_1 \rrbracket_{TESL} \geq \text{suc } n_1] \rangle_{config} \\
\geq \langle [\llbracket \Gamma_2 \rrbracket_{prim} \cap [\llbracket \Psi_2 \rrbracket_{TESL} \geq n_2] \cap [\llbracket \Phi_2 \rrbracket_{TESL} \geq \text{suc } n_2] \rangle_{config} \quad \text{(is ?P)} \\
\text{proof -} \\
\]

47
CHAPTER 7. MAIN THEOREMS

from assms consider

(a) \((\Gamma_1, n_1 \vdash \psi_1 \supset \phi_1) \leadsto (\Gamma_2, n_2 \vdash \psi_2 \supset \phi_2)\)

(b) \((\Gamma_1, n_1 \vdash \psi_1 \supset \phi_1) \leadsto (\Gamma_2, n_2 \vdash \psi_2 \supset \phi_2)\)

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 \(\Gamma \vdash K \Psi \Phi\)

assume \((\Gamma_1, n_1 \vdash \psi_1 \supset \phi_1) = (\Gamma, n \vdash (K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi \supset \Phi)\)

and \((\Gamma_2, n_2 \vdash \psi_2 \supset \phi_2) = (\Gamma, n \vdash \Psi \supset ((K_2 \text{ sporadic } \tau \text{ on } K_2) \# \Phi))\)

thus \(?P using HeronConf_interp_stepwise_sporadic_cases

HeronConf_interpretation.simps by blast

next

fix \(\Gamma \vdash K_1 \supset K_2 \Psi \Phi\)

assume \((\Gamma_1, n_1 \vdash \psi_1 \supset \phi_1) = (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \supset \Phi)\)

and \((\Gamma_2, n_2 \vdash \psi_2 \supset \phi_2) = (\Gamma, n \vdash \Psi \supset ((K_1 \text{ implies } K_2) \# \Phi))\)

thus \(?P using HeronConf_interp_stepwise_sporadic_cases

HeronConf_interpretation.simps by blast

next

fix \(\Gamma \vdash K_1 \supset K_2 \Psi \Phi\)

assume \((\Gamma_1, n_1 \vdash \psi_1 \supset \phi_1) = (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \supset \Phi)\)

and \((\Gamma_2, n_2 \vdash \psi_2 \supset \phi_2) = (\Gamma, n \vdash \Psi \supset ((K_1 \text{ implies } K_2) \# \Phi))\)

thus \(?P using HeronConf_interp_stepwise_sporadic_cases

HeronConf_interpretation.simps by blast

next

fix \(\Gamma \vdash K_1 \supset K_2 \Psi \Phi\)

assume \((\Gamma_1, n_1 \vdash \psi_1 \supset \phi_1) = (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \supset \Phi)\)

and \((\Gamma_2, n_2 \vdash \psi_2 \supset \phi_2) = (\Gamma, n \vdash \Psi \supset ((K_1 \text{ implies } K_2) \# \Phi))\)

thus \(?P using HeronConf_interp_stepwise_sporadic_cases

HeronConf_interpretation.simps by blast

next

fix \(\Gamma \vdash K_1 \supset K_2 \Psi \Phi\)

assume \((\Gamma_1, n_1 \vdash \psi_1 \supset \phi_1) = (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \supset \Phi)\)

and \((\Gamma_2, n_2 \vdash \psi_2 \supset \phi_2) = (\Gamma, n \vdash \Psi \supset ((K_1 \text{ implies } K_2) \# \Phi))\)

thus \(?P using HeronConf_interp_stepwise_sporadic_cases

HeronConf_interpretation.simps by blast

next

fix \(\Gamma \vdash K_1 \supset K_2 \Psi \Phi\)

assume \((\Gamma_1, n_1 \vdash \psi_1 \supset \phi_1) = (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \supset \Phi)\)

and \((\Gamma_2, n_2 \vdash \psi_2 \supset \phi_2) = (\Gamma, n \vdash \Psi \supset ((K_1 \text{ implies } K_2) \# \Phi))\)

thus \(?P using HeronConf_interp_stepwise_sporadic_cases

HeronConf_interpretation.simps by blast

next

fix \(\Gamma \vdash K_1 \supset K_2 \Psi \Phi\)

assume \((\Gamma_1, n_1 \vdash \psi_1 \supset \phi_1) = (\Gamma, n \vdash (K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi \supset \Phi)\)
and \( (\Gamma_2, n_2 \vdash \Psi_2 \bowtie \Phi_2) = \)
\( (((K_1 \bowleft n) \# \Gamma), n \vdash \Psi \bowtie ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi)) \)

thus ?P using HeronConf_interp_stepwise_timedelayed_cases
HeronConf_interpretation.simps by blast

next
fix \( \Gamma \vdash \Gamma_2, K_2, K_3, \Psi, \Phi \)
assume \( (\Gamma, n \vdash (((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \bowtie \Phi)) \)
and \( (\Gamma_2, n_2 \vdash \Psi_2 \bowtie \Phi_2) = \)
\( (((K_1 \bowleft n) \# (K_2 \bowleft n) \# \delta \tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \bowtie ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi)) \)

thus ?P using HeronConf_interp_stepwise_timedelayed_cases
HeronConf_interpretation.simps by blast

next
fix \( \Gamma \vdash \Gamma_2, K_2, K_3, \Psi, \Phi \)
assume \( (\Gamma_1, n_1 \vdash \Psi_1 \bowtie \Phi_1) = \)
\( (\Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \bowtie \Phi)) \)
and \( (\Gamma_2, n_2 \vdash \Psi_2 \bowtie \Phi_2) = \)
\( (((K_1 \bowleft n) \# (K_2 \bowleft n) \# \Psi \bowtie ((\lambda(x, y). x \leq y) \# \Gamma), n \vdash \Psi \bowtie ((K_1 \text{ weakly precedes } K_2) \# \Phi)) \)

next
fix \( \Gamma \vdash \Gamma_2, K_2, K_3, \Psi, \Phi \)
assume \( (\Gamma_1, n_1 \vdash \Psi_1 \bowtie \Phi_1) = \)
\( (\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \bowtie \Phi)) \)
and \( (\Gamma_2, n_2 \vdash \Psi_2 \bowtie \Phi_2) = \)
\( (((K_1 \bowleft n) \# (K_2 \bowleft n) \# \Psi \bowtie ((\lambda(x, y). x \leq y) \# \Gamma), n \vdash \Psi \bowtie ((K_1 \text{ strictly precedes } K_2) \# \Phi)) \)

next
fix \( \Gamma \vdash \Gamma_2, K_2, K_3, \Psi, \Phi \)
assume \( (\Gamma_1, n_1 \vdash \Psi_1 \bowtie \Phi_1) = \)
\( (\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \bowtie \Phi)) \)
and \( (\Gamma_2, n_2 \vdash \Psi_2 \bowtie \Phi_2) = \)
\( (((K_1 \bowleft n) \# (K_2 \bowleft n) \# \Psi \bowtie ((\lambda(x, y). x \leq y) \# \Gamma), n \vdash \Psi \bowtie ((K_1 \text{ kills } K_2) \# \Phi)) \)

qed

inductive_cases step_elim:\((S_1 \hookrightarrow S_2)\)

lemma sound_reduction':
assumes \((S_1 \hookrightarrow S_2)\)
shows \([S_1]_{config} \sqsupseteq [S_2]_{config}\)
proof -
have \(\forall s_1, s_2, \langle s_2 \rangle_{config} \subseteq [s_1]_{config} \vee \neg(s_1 \hookrightarrow s_2)\)
using sound_reduction by fastforce
thus ?thesis using assms by blast
qed

lemma sound_reduction_generalized:
assumes \((S_1 \hookrightarrow^* S_2)\)
shows \([S_1]_{config} \sqsupseteq [S_2]_{config}\)
proof -
from assms show ?thesis
proof (induction k arbitrary: S)
  case 0
  hence *: (S1 ⊢^0 S2 ==> S1 = S2) by auto
  moreover have (S1 = S2) using * by linarith
  ultimately show ?case by auto
next
  case (Suc k)
  thus ?case
  proof -
    fix k :: nat
    assume ff: (S1 ⊢^Suc k S2)
    assume hi: (⋀S2. S1 ⊢^k S2 ==> S1 ⚬ config ⊆ S1 ⚬ config)
    obtain Sn where red_decomp: (S1 ⊢^k Sn) ∧ (Sn ⊢^k S2) using ff by auto
    hence (S1 ⚬ config ⊇ Sn ⚬ config) using hi by simp
    also have (Sn ⚬ config ⊇ S2 ⚬ config) by (simp add: red_decomp sound_reduction')
    ultimately show (S1 ⚬ config ⊇ S2 ⚬ config) by simp
  qed
qed

From the initial configuration, a configuration S obtained after any number k of reduction steps

denotes runs from the initial specification Ψ.

theorem soundness:
assumes ⟨([], 0 ⊢ Ψ ⊢ []) ⊢^k S⟩
shows ⟨[[Ψ] T ESL ⊇ [[S] config]⟩
using assms sound_reduction_generalized solve_start by blast

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.

lemma complete_direct_successors:
shows ⟨Γ, n ⊢ Ψ ⊢ Φ ⚬ config ⊆ ∪X ∈ Cnext (Γ, n ⊢ Ψ ⊢ Φ ⚬ X ⚬ config)⟩
proof (induct Ψ)
  case Nil
  show ?case
    using HeronConf_interp_stepwise_instant_cases operational_semantics_step.simps
    operational_semantics_intro.instant_i
    by fastforce
next
  case (Cons ψ Ψ)
  thus ?case
  proof (cases ψ)
    case (SporadicOn K1 τ K2)
    thus ?thesis
    using HeronConf_interp_stepwise_sporadicon_cases
    [of (Γ) m (K1) (τ) (K2) (ψ) (Φ)]
    Cnext_solve_sporadicon[of (Γ) m (Ψ) (K1) (τ) (K2) (Φ) ] by blast
next
  case (TagRelation K1 K2 R)
  thus ?thesis
  using HeronConf_interp_stepwise_tagrel_cases
  [of (Γ) m (K1) (K2) R (ψ) (Φ)]
  Cnext_solve_tagrel[of (K1) m (K2) R (Γ) (ψ) (Φ) ] by blast
next
  case (Implies K1 K2)
  thus ?thesis

7.3. **Completeness**

using HeronConf_interp_stepwise_implies_cases

[\{of (\Gamma) \n \} \{K1) \{K2) (\Psi) (\Phi)\]

Cnext_solve_implies[of (K1) \n \} \{\Gamma) \{\Psi) \{K2) (\Phi)\] by blast

next

case (ImpliesNot K1 K2) thus ?thesis

using HeronConf_interp_stepwise_implies_not_cases

[\{of (\Gamma) \n \} \{K1) \{K2) (\Psi) (\Phi)\]

Cnext_solve_implies_not[of (K1) \n \} \{\Gamma) \{\Psi) \{K2) (\Phi)\] by blast

next

case (TimeDelayedBy Kmast \tau Kmeas Kslave) thus ?thesis

using HeronConf_interp_stepwise_timedelayed_cases

[\{of (\Gamma) \n \} \{Kmast) \{\tau) \{Kmeas) \{Kslave) (\Psi) (\Phi)\]

Cnext_solve_timedelayed

[\{of \{Kmast) \n \} \{\Gamma) \{\Psi) \{\tau) \{Kmeas) \{Kslave) (\Phi)\] by blast

next

case (WeaklyPrecedes K1 K2) thus ?thesis

using HeronConf_interp_stepwise_weakly_precedes_cases

[\{of (\Gamma) \n \} \{K1) \{K2) (\Psi) (\Phi)\]

Cnext_solve_weakly_precedes[of (K2) \n \} \{\Gamma) \{\Psi) \{K1) (\Phi)\] by blast

next

case (StrictlyPrecedes K1 K2) thus ?thesis

using HeronConf_interp_stepwise_strictly_precedes_cases

[\{of (\Gamma) \n \} \{K1) \{K2) (\Psi) (\Phi)\]

Cnext_solve_strictly_precedes[of (K2) \n \} \{\Gamma) \{\Psi) \{K1) (\Phi)\] by blast

next

case (Kills K1 K2) thus ?thesis

using HeronConf_interp_stepwise_kills_cases[of (\Gamma) \n \} \{K1) \{K2) (\Psi) (\Phi)\]

Cnext_solve_kills[of \{K1) \n \} \{\Gamma) \{\Psi) \{K2) (\Phi)\] by blast

qed

lemma complete_direct_successors:

shows \{\mathcal{S} \in config \subset (\bigcup X \in C_{next} \mathcal{S}, \[X \in config)\)

proof -

from HeronConf_interpretation_cases obtain \Gamma \ n \ \Psi \ \Phi

where \mathcal{S} = (\Gamma, \ n \tau \ \Psi \rightarrow \ \Phi) by blast

with complete_direct_successors[of (\Gamma) \n \} \{\Psi) (\Phi)\] show ?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 \(\exists S_1 \in \mathcal{S}_1 \in config\)

shows \(\exists S_2. (S_1 \rightarrow S_2) \land (\rho \in \mathcal{S}_2 \in config)\)

proof -

from assms complete_direct_successors have \(\rho \in \bigcup X \in C_{next} \mathcal{S}_1, \[X \in config)\) by blast

hence \(\exists \mathcal{S} \in config \mathcal{S}_1, \ \rho \in \mathcal{S} \in config) by simp

thus ?thesis by blast

qed

lemma branch_existence':

assumes \(\rho \in \mathcal{S}_1 \in config\)

shows \(\exists S_2. (S_1 \rightarrow S_2) \land (\rho \in \mathcal{S}_2 \in config)\)

proof (induct k)

case 0

thus ?case by (simp add: assms)
next
  case (Suc k)
  then show ?case
  using branch_existence relopwp_Suc_I[of k] (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 \( \langle \varrho \in \{ \{ \Psi \} \} \rangle \)
shows \( \exists S. \ (\langle \varnothing, 0 \triangleright \varnothing \rangle \rightsquigarrow^k S) \)
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 \( \langle \varrho \in \Gamma, n \triangleright \Psi \triangleright \Phi \rangle \)
shows \( \langle \exists \Gamma_k \Psi_k \Phi_k. \ (\Gamma, n \triangleright \Psi \triangleright \Phi) \rightsquigarrow^k (\Gamma_k, \text{Suc} n \triangleright \Psi_k \triangleright \Phi_k) \rangle \)
proof (insert assms, induct \( \Psi \) arbitrary: \( \Gamma \ Phi \))
  case (Nil \( \Gamma \) \( \Phi \))
  then show ?case
  proof 
    have \( \langle \Gamma, n \triangleright \emptyset \triangleright \emptyset \rangle \)
      using \( \text{instant}_i \) intro_part by fastforce
    moreover have \( \langle \Gamma, \text{Suc} n \triangleright \emptyset \triangleright \emptyset \rangle \)
      by auto
    moreover have \( \langle \varrho \in \Gamma, \text{Suc} n \triangleright \emptyset \triangleright \emptyset \rangle \)
      using assms Nil.prems calculation(2) by blast
    ultimately show ?thesis by blast
  qed
next
  case (Cons \( \psi \) \( \Psi \))
  then show ?case
  proof (induct \( \psi \))
    case (SporadicOn \( K_1 \) \( \tau \) \( K_2 \))
    have branches: \( \langle \Gamma, \text{Suc} n \triangleright \emptyset \triangleright \emptyset \rangle \)
      \( \Rightarrow \exists \Gamma_k \Psi_k \Phi_k. \)
      \( \langle (K_1 \triangleright \tau \text{ on } K_2) \# \Psi \rangle \triangleright \Phi \)
      \( \rightsquigarrow^k (\Gamma_k, \text{Suc} n \triangleright \Psi_k \triangleright \Phi_k) \)
      \( \wedge \varrho \in \Gamma_k, \text{Suc} n \triangleright \Psi_k \triangleright \Phi_k \rangle \)
      using HeronConf_interop_stepwise_sporadic_cases by simp
    have br1: \( \langle \varrho \in \Gamma, \text{Suc} n \triangleright \psi \triangleright ((K_1 \triangleright \tau \text{ on } K_2) \# \Psi) \rangle \)
      \( \Rightarrow \exists \Gamma_k \Psi_k \Phi_k. \)
      \( \langle (K_1 \triangleright \tau \text{ on } K_2) \# \Psi \rangle \triangleright \Phi \)
      \( \rightsquigarrow^k (\Gamma_k, \text{Suc} n \triangleright \Psi_k \triangleright \Phi_k) \)
      \( \wedge \varrho \in \Gamma_k, \text{Suc} n \triangleright \Psi_k \triangleright \Phi_k \rangle \)
      proof
        assume br1: \( \langle \varrho \in \Gamma, \text{Suc} n \triangleright \psi \triangleright ((K_1 \triangleright \tau \text{ on } K_2) \# \Psi) \rangle \)
        hence \( \exists \Gamma_k \Psi_k \Phi_k. \)
        \( \langle (K_1 \triangleright \tau \text{ on } K_2) \# \Psi \rangle \triangleright \Phi \)
        \( \rightsquigarrow^k (\Gamma_k, \text{Suc} n \triangleright \Psi_k \triangleright \Phi_k) \)
        \( \wedge \varrho \in \Gamma_k, \text{Suc} n \triangleright \Psi_k \triangleright \Phi_k \rangle \)
      qed
using h1 SporadicOn.prems by simp
from this obtain Γₖ ψₖ φₖ k
where
    fp := (\( (\Gamma', n \vdash \psi \triangleright ((K₁ sporadic \tau on K₂) \# \phi) ) \rightarrow^k (\Gammaₖ', Suc n \vdash \psiₖ \triangleright \phiₖ) \))
    \& \( \rho \in \{ Γₖ', Suc n \vdash \psiₖ \triangleright \phiₖ \}_{\text{config}} \) by blast
have
    \( (\Gamma', n \vdash ((K₁ sporadic \tau on K₂) \# \psi) \triangleright \phi) \rightarrow (\Gamma, n \vdash ((K₁ sporadic \tau on K₂) \# \psi) \triangleright \phi) \)
by (simp add: elims_part sporadic_on_el)
with fp relpowp_Suc_I2 have
    \( (\Gamma', n \vdash ((K₁ sporadic \tau on K₂) \# \psi) \triangleright \phi) \rightarrow^suc k (\Gammaₖ', Suc n \vdash \psiₖ \triangleright \phiₖ) \)
thus \( \text{thesis} \) using fp by blast
qed

have br2: \( \rho \in \{ ((K₁ \uparrow n) \# (K₂ \downarrow n \& \tau) \# \Gamma), n \vdash \psi \triangleright \phi \}_{\text{config}} \implies \exists Γₖ ψₖ φₖ k.
(\( (\Gamma', n \vdash ((K₁ sporadic \tau on K₂) \# \psi) \triangleright \phi) \rightarrow^k (\Gammaₖ', Suc n \vdash \psiₖ \triangleright \phiₖ) \))
\& \( \rho \in \{ Γₖ', Suc n \vdash \psiₖ \triangleright \phiₖ \}_{\text{config}} \)
proof -
assume h2: \( \rho \in \{ ((K₁ \uparrow n) \# (K₂ \downarrow n \& \tau) \# \Gamma), n \vdash \psi \triangleright \phi \}_{\text{config}} \)
then \( \exists Γₖ ψₖ φₖ k.
(\( (\Gamma', n \vdash ((K₁ sporadic \tau on K₂) \# \psi) \triangleright \phi) \rightarrow^k (\Gammaₖ', Suc n \vdash \psiₖ \triangleright \phiₖ) \))
\& \( \rho \in \{ Γₖ', Suc n \vdash \psiₖ \triangleright \phiₖ \}_{\text{config}} \)
using h2 SporadicOn.prems by simp
from this obtain Γₖ ψₖ φₖ k
where
    fp := (\( (\Gamma', n \vdash \psi \triangleright ((K₁ sporadic \tau on K₂) \# \phi) ) \rightarrow^k (\Gammaₖ', Suc n \vdash \psiₖ \triangleright \phiₖ) \))
    \& \( \rho \in \{ Γₖ', Suc n \vdash \psiₖ \triangleright \phiₖ \}_{\text{config}} \) by blast
and rc := (\( \rho \in \{ Γₖ', Suc n \vdash \psiₖ \triangleright \phiₖ \}_{\text{config}} \) by blast
have pc := (\( (\Gamma', n \vdash ((K₁ sporadic \tau on K₂) \# \psi) \triangleright \phi) \rightarrow (\Gamma, n \vdash ((K₁ sporadic \tau on K₂) \# \psi) \triangleright \phi) \))
by (simp add: elims_part sporadic_on_el)

hence (\( (\Gamma', n \vdash ((K₁ sporadic \tau on K₂) \# \psi) \triangleright \phi) \rightarrow^k (\Gammaₖ', Suc n \vdash \psiₖ \triangleright \phiₖ) \))
\& \( \rho \in \{ Γₖ', Suc n \vdash \psiₖ \triangleright \phiₖ \}_{\text{config}} \)
using fp relpowp_Suc_I2 by auto
with rc show \( \text{thesis} \) by blast
qed

from branches SporadicOn.prems(2) have
    \( \rho \in \{ \Gamma, n \vdash \psi \triangleright ((K₁ sporadic \tau on K₂) \# \phi) \}_{\text{config}} \)
\& \( \{ ((K₁ \uparrow n) \# (K₂ \downarrow n \& \tau) \# \Gamma), n \vdash \psi \triangleright \phi \}_{\text{config}} \)
by simp
with br1 br2 show \( \text{case} \) by blast
next

\text{case (TagRelation K₁ K₂ R)}

have branches: \( \{ \Gamma, n \vdash ((\text{time-relation } [K₁, K₂] \in R) \# \psi) \triangleright \phi \}_{\text{config}} \)
\& \( \{ ([\text{time-relation } [K₁, K₂] \in R) \# \psi) \}_{\text{config}} \)
using HeronConf_intermp_stepwise_tagrel_cases by simp
thus \( \text{case} \) proof -

have (\( \exists Γₖ ψₖ φₖ k.
((\( ([\text{time-relation } [K₁, K₂] \in R) \# \psi) \rightarrow^k (\Gammaₖ', Suc n \vdash \psiₖ \triangleright \phiₖ) \))\) \& \( \rho \in \{ Γₖ', Suc n \vdash \psiₖ \triangleright \phiₖ \}_{\text{config}} \))
using TagRelation.prems by simp
from this obtain Γₖ ψₖ φₖ k
where
    fp := (\( ((\( ([\text{time-relation } [K₁, K₂] \in R) \# \psi) \rightarrow (\Gamma', n \vdash ((([\text{time-relation } [K₁, K₂] \in R) \# \psi) ) \rightarrow^k (\Gammaₖ', Suc n \vdash \psiₖ \triangleright \phiₖ) \)) \& \( \rho \in \{ Γₖ', Suc n \vdash \psiₖ \triangleright \phiₖ \}_{\text{config}} \))

\[ \omega^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \trianglelefteq \Phi_k) \]
and rec: \(q \in [ \Gamma_k, \text{Suc } n \vdash \Psi_k \trianglelefteq \Phi_k]_{\text{config}} \) by blast
have pc: ((time-relation [K, K2] \in R) \# \Psi \triangleright \Phi)
\[ \vdash ((\tau_{\text{var}} (K, n), \tau_{\text{var}} (K2, n)) \in R) \# \Gamma, n \vdash \Psi \triangleright ((\text{time-relation} [K, K2] \in R) \# \Phi)) \]
by (simp add: elims_part tagrel_e)
hence \((\Gamma, n \vdash (\text{time-relation} [K, K2] \in R) \# \Psi \triangleright \Phi)\)
\[ \vdash_{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \trianglelefteq \Phi_k) \)
using fp relpowp_Suc_12 by auto
with rc show ?thesis by blast
qed
next
case (Implies K1, K2)
  have branches: [\[ \Gamma, n \vdash ((K1 \text{ implies } K2) \# \Psi) \triangleright \Phi ]_{\text{config}}
  = [ ((K1 \neg \triangleright n) \# \Gamma), n \vdash \Psi \triangleright ((K1 \text{ implies } K2) \# \Phi) ]_{\text{config}}
  \cup [ ((K1 \triangleright n) \# (K2 \triangleright n) \# \Gamma), n \vdash \Psi \triangleright ((K1 \text{ implies } K2) \# \Phi) ]_{\text{config}}
  using HeronConf_interstepwise_implies_cases by simp
moreover have br1: \(q \in [ ((K1 \neg \triangleright n) \# \Gamma), n \vdash \Psi \triangleright ((K1 \text{ implies } K2) \# \Phi) ]_{\text{config}}\)
\[ \vdash_{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \trianglelefteq \Phi_k) \]
\[ \land q \in [ \Gamma_k, \text{Suc } n \vdash \Psi_k \trianglelefteq \Phi_k ]_{\text{config}} \]
proof -
  assume h1: \(q \in [ ((K1 \neg \triangleright n) \# \Gamma), n \vdash \Psi \triangleright ((K1 \text{ implies } K2) \# \Phi) ]_{\text{config}}\)
  then have [\[ \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma, n \vdash ((K1 \text{ implies } K2) \# \Psi) \triangleright \Phi) \]
  \[ \vdash_{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]
using h1 implies.proves by simp
from this obtain \(\Gamma_k \Psi_k \Phi_k k\) where
  \[ \vdash_{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]
and rec: \(q \in [ \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k ]_{\text{config}} \) by blast
have pc: ((\Gamma, n \vdash (K1 \text{ implies } K2) \# \Psi) \triangleright \Phi)
\[ \vdash ((K1 \neg \triangleright n) \# \Gamma), n \vdash \Psi \triangleright ((K1 \text{ implies } K2) \# \Phi)) \]
by (simp add: elims_part_implies_e1)
hence \((\Gamma, n \vdash (K1 \text{ implies } K2) \# \Psi \triangleright \Phi) \)
\[ \omega^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]
using fp relpowp_Suc_12 by auto
with rc show ?thesis by blast
qed
moreover have br2: \(q \in [ ((K1 \triangleright n) \# (K2 \triangleright n) \# \Gamma), n \vdash \Psi \triangleright ((K1 \text{ implies } K2) \# \Phi) ]_{\text{config}}\)
\[ \vdash_{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]
\[ \land q \in [ \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k ]_{\text{config}} \]
proof -
  assume h2: \(q \in [ ((K1 \triangleright n) \# (K2 \triangleright n) \# \Gamma), n \vdash \Psi \triangleright ((K1 \text{ implies } K2) \# \Phi) ]_{\text{config}}\)
  then have [\[ \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma, n \vdash ((K1 \text{ implies } K2) \# \Psi) \triangleright \Phi) \]
  \[ \vdash_{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]
using h2 implies.proves by simp
from this obtain \(\Gamma_k \Psi_k \Phi_k k\) where
  \[ \vdash_{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \]
and rec: \(q \in [ \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k ]_{\text{config}} \) by blast
have \((\Gamma, n \vdash (K1 \text{ implies } K2) \# (K1 \neg \triangleright n) \# \Gamma), n \vdash \Psi \triangleright ((K1 \text{ implies } K2) \# \Phi)) \)
by (simp add: elims_part_implies_e2)
7.4. PROGRESS

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

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: 

\[
\begin{align*}
&\{ (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi \} \text{\_config} \\
&\text{\_config} \\
&\text{\_config}
\end{align*}
\]

using HeronConf\_interp\_stepwise\_implies\_not\_cases by simp
moreover have br1: \(\emptyset \in \{ (K_1 \dashv \vdash n) \# \Gamma, n \vdash \Psi \Rightarrow ((K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi) \} \text{\_config}\)
then have \(\exists \Gamma_k \Psi_k \Phi_k . ((\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi) \Rightarrow k \) \((\Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k)\)
\& \(\emptyset \in \{ \Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k \} \text{\_config}\)
using h1 Implies\_Not\_prems by simp

from this obtain \(\Gamma_k \Psi_k \Phi_k \) where
fp:\(((K_1 \dashv \vdash n) \# \Gamma, n \vdash \Psi \Rightarrow ((K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi) \Rightarrow k \) \((\Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k)\)
and rc:\(\emptyset \in \{ \Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k \} \text{\_config}\) by blast
have pc:\((\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi) \Rightarrow k \) \((\Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k)\)
by (simp add: elims\_part\_implies\_not\_el)

hence \((\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi) \Rightarrow_{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k)\)
using fp relpowp\_Suc_I2 by auto
with rc show \?thesis by blast
qed

moreover have br2: \(\emptyset \in \{ (K_1 \dashv \vdash n) \# (K_2 \dashv \vdash n) \# \Gamma, n \vdash \Psi \Rightarrow ((K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi) \} \text{\_config}\)
then have \(\exists \Gamma_k \Psi_k \Phi_k . ((\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi) \Rightarrow k \) \((\Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k)\)
\& \(\emptyset \in \{ \Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k \} \text{\_config}\)
using h2 Implies\_Not\_prems by simp

from this obtain \(\Gamma_k \Psi_k \Phi_k \) where
fp:\(((K_1 \dashv \vdash n) \# (K_2 \dashv \vdash n) \# \Gamma, n \vdash \Psi \Rightarrow ((K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi) \Rightarrow k \) \((\Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k)\)
and rc:\(\emptyset \in \{ \Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k \} \text{\_config}\) by blast
have \((\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi) \Rightarrow k \) \((\Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k)\)
by (simp add: elims\_part\_implies\_not\_el)

hence \((\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi) \Rightarrow \Phi) \Rightarrow_{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \Rightarrow \Phi_k)\)
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*}
& \text{\( (\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi \) in } \text{config} \\
& \text{\( \quad = (\{K_1 \neg\neg n\} \# \Gamma) \}, n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \) in } \text{config} \\
& \quad \cup (\{K_1 \uparrow n\} \# (K_2 \# 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) \) in } \text{config}
\end{align*} \]

using \( \text{HeronConf_interp_stepwise_timedelayed_cases by simp} \) moreover have br1:

\[ (\delta \tau \in \{K_1 \neg\neg n\} \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \) in } \text{config} \]

\[ \text{\( \Rightarrow \exists \delta \tau (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \) in } \text{config} \]

proof -

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

then have \( \exists \Gamma_k \Psi_k \Phi_k k. \)

\[ \langle \text{\( \exists \Gamma_k \Psi_k \Phi_k k. \) in } \text{config} \rangle \]

using h1 \( \text{TimeDelayedBy.prems by simp} \) from this obtain \( \Gamma_k \Psi_k \Phi_k k \)

where \( \text{fp:} \langle \text{\( \exists \Gamma_k \Psi_k \Phi_k k. \) in } \text{config} \rangle \]

and \( \text{rc:} \langle \text{\( \exists \Gamma_k \Psi_k \Phi_k k. \) in } \text{config} \rangle \) by blast

have \( \langle \text{\( \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \) in } \text{config} \rangle \)

by (simp add: elims_part_timedelayed_el)

hence \( \langle \text{\( \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \) in } \text{config} \rangle \)

using \( \text{fp relpow_Suc_I2 by auto} \) with \( \text{rc show } ? \text{thesis by blast} \)

qed moreover have br2:

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

\[ \Rightarrow \exists \Gamma_k \Psi_k \Phi_k k. \]

\[ \langle \text{\( \exists \Gamma_k \Psi_k \Phi_k k. \) in } \text{config} \rangle \]

proof -

assume h2: \( (\delta \tau \in \{K_1 \uparrow n\} \# (K_2 \# n \oplus \delta \tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \) in } \text{config} \]

then have \( \exists \Gamma_k \Psi_k \Phi_k k. \)

\[ \langle \text{\( \exists \Gamma_k \Psi_k \Phi_k k. \) in } \text{config} \rangle \]

using h2 \( \text{TimeDelayedBy.prems by simp} \) from this obtain \( \Gamma_k \Psi_k \Phi_k k \)

where \( \text{fp:} \langle \text{\( \exists \Gamma_k \Psi_k \Phi_k k. \) in } \text{config} \rangle \]

and \( \text{rc:} \langle \text{\( \exists \Gamma_k \Psi_k \Phi_k k. \) in } \text{config} \rangle \) by blast
have \((\Gamma, n \vdash (K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \vdash \Phi)\)
\[
\leadsto ((K_1 \vdash n) \# (K_2 \circ n \oplus \delta \tau = K_3) \# \Gamma), n
\vdash \Psi \vdash ((K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Phi))
\]
by (simp add: elim_part timedelayed_e2)
with fp relpow_Suc_I2 have
\((\Gamma, n \vdash (K_1 \text{ time-delayed by } \delta \tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \vdash \Phi)\)
\[
\leadsto \text{Suc } k \ (\text{Suc } n \vdash \Psi_k \circ \Phi_k)\]
by auto
with rc show ?thesis by blast
qed
ultimately show ?case using TimeDelayedBy.prems(2) by blast
next
case (WeaklyPrecedes K_1 K_2)

have \[\Gamma, n \vdash (K_1 \text{ weakly precedes } K_2) \# \Psi \circ \Phi \# \text{config} =
\]
\[
((\# K_2 n, \# K_1 n) \in (\lambda(x, y). x \leq y) \# \Gamma), n
\vdash \Psi \circ (K_1 \text{ weakly precedes } K_2) \# \Phi \# \text{config})
\]
using HeronConf_interps_stepwise_weakly_precedes_cases by simp
moreover have \((\# K_2 n, \# K_1 n) \in (\lambda(x, y). x \leq y) \# \Gamma), n
\vdash \Psi \circ (K_1 \text{ weakly precedes } K_2) \# \Phi \# \text{config}\)
\[
\leadsto (\exists \Gamma_k \Psi_k \Phi_k. ((\Gamma_k, n \vdash (K_1 \text{ weakly precedes } K_2) \# \Psi) \circ \Phi)
\]
\[
\leadsto (\text{Suc } k (\Gamma_k, \text{Suc } n \vdash \Psi_k \circ \Phi_k))\]
proof -
assume \((\# K_2 n, \# K_1 n) \in (\lambda(x, y). x \leq y) \# \Gamma), n
\vdash \Psi \circ (K_1 \text{ weakly precedes } K_2) \# \Phi \# \text{config}\)

ultimately show ?case using WeaklyPrecedes.prems(2) by blast
next
case (StrictlyPrecedes K_1 K_2)

have \[\Gamma, n \vdash (K_1 \text{ strictly precedes } K_2) \# \Psi \circ \Phi \# \text{config} =
\]
\[
((\# K_2 n, \# K_1 n) \in (\lambda(x, y). x \leq y) \# \Gamma), n
\vdash \Psi \circ (K_1 \text{ strictly precedes } K_2) \# \Phi \# \text{config})
\]
using HeronConf_interps_stepwise_strictly_precedes_cases by simp
moreover have \((\# K_2 n, \# K_1 n) \in (\lambda(x, y). x \leq y) \# \Gamma), n
\vdash \Psi \circ (K_1 \text{ strictly precedes } K_2) \# \Psi \circ \Phi \# \text{config}\)
\[
\leadsto (\exists \Gamma_k \Psi_k \Phi_k. ((\Gamma_k, n \vdash (K_1 \text{ strictly precedes } K_2) \# \Psi) \circ \Phi)
\]
\[
\leadsto (\text{Suc } k (\Gamma_k, \text{Suc } n \vdash \Psi_k \circ \Phi_k))\]
proof -
assume \((\# K_2 n, \# K_1 n) \in (\lambda(x, y). x \leq y) \# \Gamma), n
\vdash \Psi \circ (K_1 \text{ strictly precedes } K_2) \# \Phi \# \text{config}\)
hence \( \exists \Gamma, \Psi, \Phi_k. k. (\langle (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi)\)

\( \vdash (K \not\in \Psi) \wedge (K \not\in \Phi)\)

\( \langle (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

using StrictlyPrecedes.prems by simp

from this obtain \( \Gamma, \Psi, \Phi_k \)

where \( fp:\langle (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

and \( rc:\langle (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

have \( \langle (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

by (simp add: elim_part_strictly_precedes_e)

with \( fp \) relpow_Suc_I2 have \( \langle (\not\in \Gamma) \wedge (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

by auto

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

qed

ultimately show \( \exists \Gamma, \Psi, \Phi_k \)

next
case \( K \not\in \Gamma \)

have branches: \( \langle (K \not\in \Gamma) \wedge (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

\( \langle (K \not\in \Gamma) \wedge (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

using HeronConf_interp_stepwise_kills_cases by simp

moreover have \( \langle (\not\in \Gamma) \wedge (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

by (simp add: elim_part_strictly_precedes_e)

hence \( \langle (\not\in \Gamma) \wedge (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

using \( \not\in \Gamma \) Kills.prems by simp

from this obtain \( \Gamma, \Psi, \Phi_k \)

where \( fp:\langle (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

and \( rc:\langle (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

have \( \langle (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

by (simp add: elim_part_strictly_precedes_e)

moreover have \( \langle (K \not\in \Psi) \wedge (K \not\in \Phi) \rangle \wedge (\not\in \Gamma) \not\in \Psi \not\in \Phi\)

by auto

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

qed
7.4. PROGRESS

using h2 Kills.prem by simp
from this obtain Γ_k Ψ_k Φ_k k where
  \( f_p: (((K_1 \uparrow n) \# (K_2 \rightarrow n') \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \)
  \( \rightarrow \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \)
and rc: \( \exists \gamma \in \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \) by blast
have \( ((\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \)
  \( \rightarrow (((K_1 \uparrow n) \# (K_2 \rightarrow n') \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \)
by (simp add: elim_part killing_e2)
hence \( ((\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi) \) \( \in \) \( \text{Suc } k \) \( \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \)
using fp relpowp_Suc_I2 by auto
with rc show \(!\text{thesis} by blast
qed
ultimately show \(!\text{case using Kills.prem}(2) by blast
qed

lemma instant_index_increase_generalized:
assumes \( \langle n < n_k \rangle \)
assumes \( \langle \gamma \in \Gamma_k, n \vdash \Psi \triangleright \Phi \rangle \) \( \in \) \( \text{config} \)
shows \( \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma', n \vdash \Psi \triangleright \Phi) \rightarrow ((\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \)
\( \wedge \gamma \in \Gamma_k, \delta k + \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \) \( \in \) \( \text{config} \)
proof -
  obtain \( \delta k \) where \( \text{diff} \): \( n_k = \delta k + \text{Suc } n \)
  using add.commute assms(1) less_iff_Suc_add by auto
show \(!\text{thesis} by blast
proof (subst diff, subst diff, insert assms(2), dest \( \delta k \))
  case 0 thus \(!\case
  using instant_index_increase assms(2) by simp
next
  case (Suc \( \delta k \)
  have \( f_0: \gamma \in \Gamma_k, n \vdash \Psi \triangleright \Phi \) \( \in \) \( \text{config} \)
  \( \rightarrow \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \)
  \( \wedge \gamma \in \Gamma_k, \delta k + \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \) \( \in \) \( \text{config} \)
  using Suc.hyps by blast
  obtain \( \Gamma_k \Psi_k \Phi_k k \)
  where cont: \( (((\Gamma', n \vdash \Psi \triangleright \Phi) \rightarrow ((\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \)
  \( \wedge \gamma \in \Gamma_k, \delta k + \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \) \( \in \) \( \text{config} \)
  using f0 assms(1) Suc.prem by blast
  then have \( \text{fcontinue} \): \( \exists \Gamma_k' \Psi_k' \Phi_k' k' \)
  \( \rightarrow ((\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \)
  \( \wedge \gamma \in \Gamma_k', \text{Suc } (\delta k + \text{Suc } n) \vdash \Psi_k' \triangleright \Phi_k' \) \( \in \) \( \text{config} \)
  using f0 cont instant_index_increase by blast
  obtain \( \Gamma_k' \Psi_k' \Phi_k' k' \)
  where cont2: \( (((\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \)
  \( \rightarrow ((\Gamma_k', \text{Suc } (\delta k + \text{Suc } n) \vdash \Psi_k' \triangleright \Phi_k')) \)
  \( \wedge \gamma \in \Gamma_k', \text{Suc } (\delta k + \text{Suc } n) \vdash \Psi_k' \triangleright \Phi_k' \) \( \in \) \( \text{config} \)
  using Suc.prem using fcontinue cont by blast
  have \( \text{trans} \): \( \Gamma_k, n \vdash \Psi \triangleright \Phi \) \( \rightarrow (\delta k + \text{Suc } n) \vdash \Psi_k \triangleright \Phi_k \)
  using operational_semantics_trans_generalized cont cont2 by blast
  moreover have \( \text{Suc_assoc} \): \( \text{Suc } \delta k + \text{Suc } n = \text{Suc } (\delta k + \text{Suc } n) \) by arith
  ultimately show \(!\text{case} by blast
proof (subst Suc_assoc)
  show \( \exists \Gamma_k \Psi_k \Phi_k k \)
  \( \rightarrow (\Gamma_k, n \vdash \Psi \triangleright \Phi) \rightarrow ((\Gamma_k, \text{Suc } (\delta k + \text{Suc } n) \vdash \Psi_k \triangleright \Phi_k)) \)
  \( \wedge \gamma \in \Gamma_k, \text{Suc } \delta k + \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \) \( \in \) \( \text{config} \)
  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 \( \langle \varrho \in \{ \Psi \} \rangle \)\(_{TESL} \)

shows \( \exists k \Gamma_k \Psi_k \Phi_k . \ (\{ \}, 0 \vdash \Psi \rightarrow \{ \}) \hookrightarrow^k (\Gamma_k, n \vdash \Psi_k \triangleright \Phi_k) \)

\( \land \ \varrho \in \{ \Gamma_k, n \vdash \Psi_k \triangleright \Phi_k \} \)\( _{con fig} \)

**Proof -**

have \( 1: \exists \Gamma_k \Psi_k \Phi_k k. \ (\{ \}, 0 \vdash \Psi \rightarrow \{ \}) \hookrightarrow^k (\Gamma_k, 0 \vdash \Psi_k \triangleright \Phi_k) \)

\( \land \ \varrho \in \{ \Gamma_k, 0 \vdash \Psi_k \triangleright \Phi_k \} \)\( _{con fig} \)

using \( \text{assms relpowp_0_I solve_start by fastforce} \)

show \( \text{?thesis} \)

proof (cases \( n \neq 0 \))

thus \( \text{?thesis} \) using \( \text{assms relpowp_0_I solve_start by fastforce} \)

next

\( \text{case False hence pos:} n > 0 \) by simp

from \( \text{assms solve_start} \) have \( \langle \varrho \in \{ \}, 0 \vdash \Psi \rightarrow \{ \} \rangle \)\( _{con fig} \) by blast

from \( \text{instant_index_increase_generalized[OF pos this]} \) show \( \text{?thesis by blast} \)

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 :: \( \langle \tau :: \text{linordered_field} \ TESL\_formula \Rightarrow \text{nat} \rangle \) \( (\mu) \)**

where

\( \langle \mu \{ \} \rangle = (0 :: \text{nat}) \)

\( \langle \mu \ (\varrho \neq \emptyset) \rangle = (\text{case} \ \varrho \ \text{of} \)

\( \langle \text{sporadic} \ \text{on} \rangle \Rightarrow 1 + \mu \Phi \)

\( \langle \text{else} \rangle \Rightarrow 2 + \mu \Phi \rangle \)

**fun measure_interpretation_config :: \( \langle \tau :: \text{linordered_field} \ config \Rightarrow \text{nat} \rangle \) \( (\mu_{con fig}) \)**

where

\( \langle \mu_{con fig} \ (\Gamma, n \vdash \Psi \triangleright \Phi) \rangle = \mu \Psi \)

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

**lemma elimination_rules_strictly_decreasing:**

assumes \( \langle (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow \ (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) \rangle \)

shows \( \mu \Psi_1 > \mu \Psi_2! \)

using \( \text{assms by} \ (\text{auto elim: operational_semantics_elim.cases}) \)

**lemma elimination_rules_strictly_decreasing meas:**

assumes \( \langle (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow \ (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) \rangle \)

shows \( \langle \Psi_2, \Psi_1 \rangle \in \mu_{measure} \)

using \( \text{assms by} \ (\text{auto elim: operational_semantics_elim.cases}) \)

**lemma elimination_rules_strictly_decreasing meas':**

assumes \( \langle S_1 \hookrightarrow \ S_2 \rangle \)

shows \( \langle \mu_{con fig} \rangle \)

**proof -**

from \( \text{assms obtain} \ \Gamma_1 n_1 \Psi_1 \Phi_1 \) where \( p1 :: \{ S_1 = (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \} \)

using \( \text{measure_interpretation_config.cases by blast} \)

from \( \text{assms obtain} \ \Gamma_2 n_2 \Psi_2 \Phi_2 \) where \( p2 :: \{ S_2 = (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) \} \)
7.5. LOCAL TERMINATION

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 \(\mu_{\text{config}} (\Gamma_2, n_2 \vdash \Psi_2 \trianglerightPhi_2) < \mu_{\text{config}} (\Gamma_1, n_1 \vdash \Psi_1 \trianglerightPhi_1)\) 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:
\(\langle \text{wfP} (\lambda(S_1::'a::linordered_field config) S_2. (S_1 \hookrightarrow e \leftarrow S_2)) \rangle\)
proof (simp add: \(\text{wfP\_def}\))
show \(\langle \text{wf} (((S_1::'a::linordered_field config), S_2). S_1 \hookrightarrow e \leftarrow S_2) \rangle\)
proof (rule \(\text{wf\_subset}\))
have \(\text{measure } \mu_{\text{config}} = (\langle S_2, (S_1::'a::linordered_field config) \rangle, \mu_{\text{config}} S_2 < \mu_{\text{config}} S_1)\)
by (simp add: \(\text{inv\_image\_def} \ \text{less\_eq} \ \text{measure\_def}\))
thus \(\langle \langle S_1::'a::linordered_field config), S_2). S_1 \hookrightarrow e \leftarrow S_2 \rangle \subseteq (\text{measure } \mu_{\text{config}})\)
using elimination_rules_strictly_decreasing_meas'
operational_semantics_elim_inv_def by blast
next
show \(\langle \text{wf} (\text{measure measure_interpretation_config}) \rangle\) 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.

definition dilating_fun
where
\[
\text{dilating_fun} \ (f::\text{nat} \Rightarrow \text{nat}) \ (r::'a::\text{linordered_field} \ \text{run})
\equiv 
\text{strict_mono} \ f \land (f \ 0 = 0) \land (\forall n. \ f \ n \geq n)
\land ((\exists n_0. \ f \ n_0 = n) \rightarrow (\forall c. \neg (\text{hamlet} ((\text{Rep} \_ \text{run} \ r) \ n \ c))))
\]
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 $f$

\[
\text{definition} \quad \text{dilating} \\
\text{where} \\
dilating f \text{ sub } r \equiv \text{dilating_fun } f \text{ r} \\
\land (\forall n. \text{ time } ((\text{Rep_run r}) (\text{Suc } n)) = \text{ time } ((\text{Rep_run r}) (f n) c)) \\
\land (\forall n. \text{ hamlet } ((\text{Rep_run sub}) n c) = \text{ hamlet } ((\text{Rep_run r}) (f n) c))
\]

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

\[
\text{definition} \quad \text{is_subrun} :: \langle \text{a::linordered_field run} \Rightarrow \text{a run} \Rightarrow \text{bool} \rangle \\
\text{where} \\
\text{is_subrun } \text{sub } \text{r} \equiv \text{exists f. dilating f sub r}
\]

A contracting function is the reverse of a dilating fun, 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.

\[
\text{definition} \quad \text{contracting_fun} \\
\text{where} \quad \text{contracting_fun } g \equiv \text{mono } g \land g 0 = 0 \land (\forall n. g n \leq n)
\]

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.

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

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

\[
\text{definition \(dil\_inverse\) f::(nat } \rightarrow \text{ nat } \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.

\[
\text{tick_count } r \text{ c n is the number of ticks of clock } c \text{ in run } r \text{ upto instant } n.
\]

\[
\text{definition tick_count :: } \langle \text{'}a::\text{linordered_field } \text{run } \Rightarrow \text{clock } \Rightarrow \text{nat } \Rightarrow \text{nat } \rangle
\]

\[
\text{where}
\]
\[
\langle \text{tick_count } r \text{ c n } = \text{card } (i. i \leq n \land \text{hamlet } ((\text{Rep_run } r) i c))
\]

\[
\text{tick_count_strict } r \text{ c n is the number of ticks of clock } c \text{ in run } r \text{ upto but excluding instant } n.
\]

\[
\text{definition tick_count_strict :: } \langle \text{'}a::\text{linordered_field } \text{run } \Rightarrow \text{clock } \Rightarrow \text{nat } \Rightarrow \text{nat } \rangle
\]

\[
\text{where}
\]
\[
\langle \text{tick_count_strict } r \text{ c n } = \text{card } (i. i < n \land \text{hamlet } ((\text{Rep_run } r) i c))
\]

8.1.3 Stuttering Lemmas

\[
\text{theory StutteringLemmas}
\]

\[
\text{imports StutteringDefs}
\]

\[
\text{begin}
\]

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: \( \forall k. k < \overline{m} \Rightarrow \overline{P} (\text{Suc} (z + k)) = \overline{P} z \)

Shows: \( k < \overline{m} \Rightarrow \overline{P} (\text{Suc} (z + k)) = \overline{P} z \)

**Proof (induction k):**

- Case 0
  - With assms(1)[of 0] show ?case by simp

- Next case \( \text{Suc} k' \)
  - With assms[of \( \text{Suc} k' \)] show ?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: \( \text{dilating}_{\text{fun}} f \overline{r} \)

Shows: \( \text{inj}_{\text{on}} f A \)

Using assms dilating_fun_def strict_mono_imp_inj_on by blast

**Lemma dilating_injects:**

Assumes: \( \text{dilating} f \overline{r} \)

Shows: \( \text{inj}_{\text{on}} f A \)

Using 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: \( \text{dilating}_{\text{fun}} f \overline{r} \)

And \( \text{hamlet} ((\overline{\text{Rep}}_{\overline{r}}) n c) \)

Shows: \( \exists n_0. f n_0 = n \)

Using dilating_fun_def assms by blast

**Lemma ticks_image_sub:**

Assumes: \( \text{dilating}_{\text{fun}} f \overline{r} \)

And \( \exists c. \text{hamlet} ((\overline{\text{Rep}}_{\overline{r}}) n c) \)

Shows: \( \exists n_0. f n_0 = n \)

Using assms dilating_def ticks_image by blast

**Lemma ticks_image_sub':**

Assumes: \( \text{dilating}_{\text{fun}} f \overline{r} \)

And \( \exists c. \text{hamlet} ((\overline{\text{Rep}}_{\overline{r}}) n c) \)

Shows: \( \exists n_0. f n_0 = n \)

Using ticks_image_sub[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: \( \text{dilating}_{\text{fun}} f \overline{r} \)

Shows: \( \forall k. f m < k \land k < f n \land \text{hamlet} ((\overline{\text{Rep}}_{\overline{r}}) k c) \Rightarrow \overline{\text{image}} f \{k. f m < k \land k < f n \land \text{hamlet} ((\overline{\text{Rep}}_{\overline{r}}) (f k) c)\} \)

Using ?IMG = image f ?SET by blast

**Proof:**

- Fix \( k \) assume \( h: k \in ?IMG \)
  - From \( h \) obtain \( k_0 \) where \( \overline{\text{prop}}: f k_0 = k \land \text{hamlet} ((\overline{\text{Rep}}_{\overline{r}}) (f k_0) c) \)
8.1. STUTTERING INVARIANCE

using ticks_image[OF assms] by blast
with h have \( k \in \text{image } f \ ?\SET \).
using assms dilating_fun_def strict_mono_less by blast
} thus \( \?IMG \subseteq \text{image } f \ ?\SET \) ..
next
{ fix k assume h: \( k \in \text{image } f \ ?\SET \).
from h obtain \( k_0 \) where k0prop: \( k = f k_0 \land k_0 \in \?\SET \) by blast
hence \( k \in \?IMG \) using assms by (simp add: dilating_fun_def strict_mono_less)
} thus \( \text{image } f \ ?\SET \subseteq \?IMG \) ..
qed

lemma dilating_fun_image_left:
assumes \( \text{dilating } f \ ?\SET \)
shows \( \{ k. f m \leq k \land k < f n \land \text{hamlet } ((\text{Rep } r) k c) \} = \text{image } f \{ k. m \leq k \land k < n \land \text{hamlet } ((\text{Rep } r) (f k) c) \} \)
(is \( \?IMG = \text{image } f \ ?\SET \))
proof
{ fix k assume h: \( k \in \?IMG \).
from h obtain \( k_0 \) where k0prop: \( f k_0 = k \land \text{hamlet } ((\text{Rep } r) (f k_0) c) \) using ticks_image[OF assms] by blast
with h have \( k \in \text{image } f \ ?\SET \).
using assms dilating_fun_def strict_mono_less strict_mono_less_eq by fastforce
} thus \( \?IMG \subseteq \text{image } f \ ?\SET \) ..
next
{ fix k assume h: \( k \in \text{image } f \ ?\SET \).
from h obtain \( k_0 \) where k0prop: \( f k_0 = k \land k_0 \in \?\SET \) by blast
hence \( k \in \?IMG \) using assms dilating_fun_def strict_mono_less strict_mono_less_eq by fastforce
} thus \( \text{image } f \ ?\SET \subseteq \?IMG \) ..
qed

lemma dilating_fun_image_right:
assumes \( \text{dilating } f \ ?\SET \)
shows \( \{ k. f m < k \land k \leq f n \land \text{hamlet } ((\text{Rep } r) k c) \} = \text{image } f \{ k. m < k \land k \leq n \land \text{hamlet } ((\text{Rep } r) (f k) c) \} \)
(is \( \?IMG = \text{image } f \ ?\SET \))
proof
{ fix k assume h: \( k \in \?IMG \).
from h obtain \( k_0 \) where k0prop: \( f k_0 = k \land \text{hamlet } ((\text{Rep } r) (f k_0) c) \) using ticks_image[OF assms] by blast
with h have \( k \in \text{image } f \ ?\SET \).
using assms dilating_fun_def strict_mono_less strict_mono_less_eq by fastforce
} thus \( \?IMG \subseteq \text{image } f \ ?\SET \) ..
next
{ fix k assume h: \( k \in \text{image } f \ ?\SET \).
from h obtain \( k_0 \) where k0prop: \( f k_0 = k \land k_0 \in \?\SET \) by blast
hence \( k \in \?IMG \) using assms dilating_fun_def strict_mono_less strict_mono_less_eq by fastforce
} thus \( \text{image } f \ ?\SET \subseteq \?IMG \) ..
qed

lemma dilating_fun_image:
assumes \( \text{dilating } f \ ?\SET \)
shows \( \{ k. f m \leq k \land k \leq f n \land \text{hamlet } ((\text{Rep } r) k c) \} = \text{image } f \{ k. m \leq k \land k \leq n \land \text{hamlet } ((\text{Rep } r) (f k) c) \} \)
(is \( \?IMG = \text{image } f \ ?\SET \))
proof
{ fix k assume h: \( k \in \?IMG \).
from h obtain \( k_0 \) where k0prop: \( f k_0 = k \land \text{hamlet } ((\text{Rep } r) (f k_0) c) \)

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

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

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

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

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

proof -
from dilating_fun_injects[OF assms] 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 assms] 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⟩)
s shows (⟨∃ k0. f k0 = k ∧ time ((Rep_run sub) k0 c) = τ⟩)
proof -
from ticks_image_sub'[OF assms(1,2)] have (⟨∃ k0. f k0 = k⟩).
from this obtain k0 where (⟨f k0 = k⟩) by blast
moreover with assms(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⟩)
s shows (⟨hamlet ((Rep_run sub) n a) = hamlet ((Rep_run r) (f n) a)⟩)
using assms by (simp add: dilating_def)

lemma no_tick_sub:
assumes (⟨dilating f sub r⟩)
s shows ( applicationContext
(∃ n0. f n0 = n) −→ ¬ hamlet ((Rep_run r) n a))
using assms 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⟩)
s shows (⟨image f {k. hamlet ((Rep_run r) k c)} = image f {k. hamlet ((Rep_run sub) k c)}⟩)
(is (⟨?R = image f ?S⟩))
proof -
{ fix k assume h: (k ∈ ?R)
with no_tick_sub[OF assms] have (⟨∃ k0. f k0 = k⟩) by blast
from this obtain k0 where (⟨k0prop: f k0 = k⟩) by blast
with ticks_sub[OF assms] h have (⟨time ((Rep_run sub) k0 c) = τ⟩) by blast
with kprop 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 ticks_sub by blast

thus \( \{\text{image } f ?S \subseteq ?R\} \) by blast

qed

Strictly monotonous functions preserve the least element.

lemma Least_strict_mono:
  assumes \( \langle \text{strict_mono } f \rangle \) and \( \langle \exists x \in S. \forall y \in S. x \leq y \rangle \)
  shows \( \langle \text{LEAST } y. y \in f ' S \rangle = f (\text{LEAST } x. x \in S) \rangle \)
  using Least_mono[OF strict_mono_mono, OF assms].

A non empty set of \( \text{nats} \) has a least element.

lemma Least_nat_ex:
  \( \langle (n::\text{nats}) \in S \Rightarrow \exists x \in S. (\forall y \in S. x \leq y) \rangle \)
  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.

lemma Least_sub:
  assumes \( \langle \text{dilating } f \text{ sub } r \rangle \) and \( \langle \exists k::\text{nats}. \text{hamlet ((Rep_run r) k c)} \rangle \)
  shows \( \langle \text{LEAST } k. k \in \{t. \text{hamlet ((Rep_run sub) t c)}\} \rangle = f (\text{LEAST } k. k \in ?R) \rangle \)
  (is \( \langle \text{LEAST } k. k \in ?R \rangle = f (\text{LEAST } k. k \in ?S) \rangle \)
  proof -
    from assms(2) have \( \exists x. x \in ?S \) by simp
    hence least: \( \exists x \in ?S. \forall y \in ?S. x \leq y \)
      using Least_nat_ex ..
    from assms(1) have \( \langle \text{strict_mono } f \rangle \) by (simp add: dilating_def dilating_fun_def)
    from Least_strict_mono[OF this least] have
      \( \langle \text{LEAST } y. y \in f ' ?S \rangle = f (\text{LEAST } k. k \in ?S) \rangle \)
      (is \( \langle \text{LEAST } k. k \in ?R \rangle = f (\text{LEAST } k. k \in ?S) \rangle \)
      with 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.

lemma ticks_imp_ticks_sub:
  assumes \( \langle \text{dilating } f \text{ sub } r \rangle \) and \( \langle \text{hamlet ((Rep_run r) k c)} \rangle \)
  shows \( \langle \exists k0. f k0 = k \land \text{hamlet ((Rep_run sub) k0 c)} \rangle \)
  proof -
    from no_tick_sub[OF assms(1)] assms(2) have \( \exists k0. f k0 = k \) by blast
    from this obtain k0 where \( f k0 = k \) by blast
    moreover with ticks_sub[OF assms(1)] assms(2)
    have \( \text{hamlet ((Rep_run sub) k0 c)} \) by blast
    ultimately show \(?thesis\) by blast
  qed

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

lemma ticks_imp_ticks_subk:
  assumes \( \langle \text{dilating } f \text{ sub } r \rangle \) and \( \langle \text{hamlet ((Rep_run r) k c)} \rangle \)
  shows \( \langle \exists k0. f k0 = k \land \text{hamlet ((Rep_run sub) k0 c)} \rangle \)
  proof -
    from no_tick_sub[OF assms(1)] assms(2) have \( \exists k0. f k0 = k \) by blast
    from this obtain k0 where \( f k0 = k \) by blast
    moreover with ticks_sub[OF assms(1)] assms(2)
    have \( \text{hamlet ((Rep_run sub) k0 c)} \) by blast
    ultimately show \(?thesis\) by blast
  qed
A dilating function preserves the tick count on an interval for any clock.

**Lemma dilated_ticks_strict:**

**Assumes:** (dilating \(f\) \(\text{sub}\) \(r\))

**Shows:**

\[
\{i. \, f \, m < i \land i < f \, n \land \text{hamlet} ((\text{Rep\_run} \, r) \, i \, c)\}
= \text{image} \, f \, \{i. \, m < i \land i < n \land \text{hamlet} ((\text{Rep\_run} \, \text{sub}) \, i \, c)\}
\]

(is \(\text{RUN} = \text{image} \, f \, \text{?SUB}\))

**Proof**

1. Fix \(i\) assume \(h: (i \in \text{?SUB})\)
   - Hence \(m < i \land i < n\) by simp
   - Hence \(f \, m < f \, i \land f \, i < (f \, n)\) using assms
     - By (simp add: dilating_def dilating_fun_def strict_monoD strict_mono_less_eq)
     - Hence \(\text{hamlet} ((\text{Rep\_run} \, r) \, (f \, i) \, c)\) using ticks_sub[OF assms] by blast
     - Ultimately have \(f \, i \in \text{?RUN}\) by simp

2. Thus \((\text{image} \, f \, \text{?SUB} \subseteq \text{?RUN})\) by blast

**Lemma dilated_ticks_left:**

**Assumes:** (dilating \(f\) \(\text{sub}\) \(r\))

**Shows:**

\[
\{i. \, f \, m \leq i \land i < f \, n \land \text{hamlet} ((\text{Rep\_run} \, r) \, i \, c)\}
= \text{image} \, f \, \{i. \, m \leq i \land i < n \land \text{hamlet} ((\text{Rep\_run} \, \text{sub}) \, i \, c)\}
\]

(is \(\text{RUN} = \text{image} \, f \, \text{?SUB}\))

**Proof**

1. 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 \(i_0\) where \(10\text{prop}: f \, i_0 = i \land \text{hamlet} ((\text{Rep\_run} \, \text{sub}) \, i_0 \, c)\) by blast
     - With \(h\) have \((f \, n < f \, i_0 \land f \, i_0 < f \, n)\) by simp
     - Moreover have \(\text{strict\_mono} \, f\) using assms dilating_def dilating_fun_def by blast
     - Ultimately have \((m \leq i_0 \land i_0 < n)\)
       - Using strict_mono_less strict_mono_less_eq by blast
   - With \(10\text{prop}\) have \(\exists \, i_0. \, f \, i_0 = i \land i_0 \in \text{?SUB}\) by blast

2. Thus \((\text{RUN} \subseteq \text{image} \, f \, \text{?SUB})\) by blast

**Lemma dilated_ticks_right:**

**Assumes:** (dilating \(f\) \(\text{sub}\) \(r\))

**Shows:**

\[
\{i. \, m \leq i \land i < f \, n \land \text{hamlet} ((\text{Rep\_run} \, r) \, i \, c)\}
= \text{image} \, f \, \{i. \, m \leq i \land i < n \land \text{hamlet} ((\text{Rep\_run} \, \text{sub}) \, i \, c)\}
\]

(is \(\text{RUN} = \text{image} \, f \, \text{?SUB}\))

**Proof**

1. 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 \(i_0\) where \(10\text{prop}: f \, i_0 = i \land \text{hamlet} ((\text{Rep\_run} \, \text{sub}) \, i_0 \, c)\) by blast
     - With \(h\) have \((f \, n \leq f \, i_0 \land f \, i_0 < f \, n)\) by simp
     - Moreover have \(\text{strict\_mono} \, f\) using assms dilating_def dilating_fun_def by blast
     - Ultimately have \((m \leq i_0 \land i_0 < n)\)
       - Using strict_mono_less strict_mono_less_eq by blast
   - With \(10\text{prop}\) have \(\exists \, i_0. \, f \, i_0 = i \land i_0 \in \text{?SUB}\) by blast

2. Thus \((\text{RUN} \subseteq \text{image} \, f \, \text{?SUB})\) by blast
proof - shows \( \langle \text{empty_dilated_prefix} \rangle \)

\[ \text{lemma} \quad \text{No tick can occur in a dilated run before the image of 0 by the dilation function.} \]

qed

next proof

\[ \text{lemma} \quad \text{dilated_ticks:} \]

qed

next

\[ \text{fix } i \quad \text{assume } h: (i \in \text{?SUB}) \]

hence \( m < i \wedge i \leq n \) by simp

hence \( f m < f i \wedge f i \leq f n \) by using assms

by (simp add: dilating_def dilating_fun_def strict_monoD strict_mono_mono_less_eq)

moreover from \( h \) have \( \text{hamlet ((Rep_run sub) i c)} \) by simp

hence \( \text{hamlet ((Rep_run r) (f i) c)} \) 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

qed

lemma dilated_ticks:

assumes (dilating f sub r)

shows \( \langle i. \ f n \leq i \wedge i \leq f n \wedge \text{hamlet ((Rep_run r) i c)} \rangle \)

= image f \( \langle i. \ m < i \wedge i \leq n \wedge \text{hamlet ((Rep_run sub) i c)} \rangle \)

(is \( \langle \text{?RUN} = \text{image f} \text{ ?SUB} \rangle \) )

proof

\{ \text{fix } i \quad \text{assume } h: (i \in \text{?SUB}) \}

hence \( m \leq i \wedge i \leq n \) by simp

hence \( f m \leq f i \wedge f i \leq (f n) \)

using assms by (simp add: dilating_def dilating_fun_def strict_mono_mono_less_eq)

moreover from \( h \) have \( \text{hamlet ((Rep_run sub) i c)} \) by simp

hence \( \text{hamlet ((Rep_run r) (f i) c)} \) 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

qed

next

\{ \text{fix } i \quad \text{assume } h: (i \in \text{?RUN}) \}

hence \( \text{hamlet ((Rep_run r) i c)} \) by simp

from ticks_imp_ticks_sub[OF assms this]

obtain \( i_0 \) where 10prop: \( f i_0 = i \wedge \text{hamlet ((Rep_run sub) i_0 c)} \) by blast

with \( h \) have \( (f n \leq f i_0 \wedge f i_0 \leq f n) \) by simp

moreover have \( \text{strict_mono f} \) using \( \text{strict_mono_less}\) by blast

ultimately have \( (m \leq i_0 \wedge i_0 \leq n) \) using \( \text{strict_mono_less}\) by blast

with 10prop have \( (\exists i_0. \ f i_0 = i \wedge i_0 \in \text{?SUB}) \) by blast

\( \) 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 (dilating f sub r)

and \( n < f 0 \)

shows \( \neg \text{hamlet ((Rep_run r) n c)} \)

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

A singleton of nat can be defined with a weaker property.

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

The set definition and the function definition of 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)
case 0
  have ⟨\ldots = card \{i. i \leq 0 \land \text{hamlet} ((\text{Rep\_run} r) i c)\}\rangle
    by (simp add: tick_count_def)
  also have ⟨\ldots = card \{i::nat. i = 0 \land \text{hamlet} ((\text{Rep\_run} r) 0 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 ⟨\ldots = (if \text{hamlet} ((\text{Rep\_run} r) 0 c) then 1 else 0)\rangle by simp
  finally show ?case .
next
case (Suc k)
  show ?case
  proof (cases \langle \text{hamlet} ((\text{Rep\_run} r) (Suc k) c) \rangle)
    case True
    hence ⟨\ldots = \text{Suc} (\text{tick\_count} r c k)\rangle
      by (simp add: tick_count_def)
    with Suc.IH have ⟨\text{tick\_count} r c (Suc k) = \text{Suc} (\text{run\_tick\_count} r c k)\rangle
      by simp
    thus ?thesis by (simp add: True)
  next
case False
    hence ⟨\ldots = \text{tick\_count} r c k\rangle
      by (simp add: tick_count_def)
    thus ?thesis using Suc.IH by (simp add: False)
  qed

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

lemma tick_count_strict_suc: \langle \text{tick\_count\_strict} r c (Suc n) = \text{tick\_count} r c n \rangle
  unfolding tick_count_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 ⟨\text{tick\_count\_strict} r c n = 0\rangle
    unfolding tick_count_strict_def by simp
  also have ⟨\ldots = \text{run\_tick\_count\_strictly} r c 0\rangle
    using run_tick_count_strictly.simps(1)[symmetric].
  finally show ?thesis using True by simp
next
case False
  from not0_implies_Suc[OF this] obtain m where \*:\langle n = Suc m \rangle by blast
  hence ⟨\text{tick\_count\_strict} r c n = \text{tick\_count} r c m\rangle
    using tick_count_strictSuc by simp
  also have ⟨\ldots = \text{run\_tick\_count\_strictly} r c m\rangle
    using run_tick_count_strictly.simps(1)[symmetric].
  also have ⟨\ldots = \text{run\_tick\_count\_strictly} r c (Suc m)\rangle
    using run_tick_count_strictly.simps(2)[symmetric].
  finally show ?thesis using * by simp
qed

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

lemma strictly_precedes_alt_def1:
  \{ g. \forall n::nat. (\text{run\_tick\_count} g K_2 n) \leq (\text{run\_tick\_count\_strictly} g K_1 n) \}
  = \{ g. \forall n::nat. (\text{run\_tick\_count\_strictly} g K_2 (Suc n)) \}
by auto

The strict precedence relation can even be defined using only \texttt{run_tick_count}:

\begin{verbatim}
lemma zero_gt_all:
  assumes \(P \equiv \emptyset\)
  and \((\forall n. \quad n > 0 \implies P\ n)\)
  shows \(P\ n\)
  using assms neq0_conv by blast
\end{verbatim}

\begin{verbatim}
lemma strictly_precedes_alt_def2:
  \((\forall \phi. \quad \forall n::nat. \quad \texttt{run_tick_count} \phi K n \leq \texttt{run_tick_count_strictly} \phi K n)\)
  \(= \quad \exists \phi. \quad \neg \text{hanlet} \quad \text{((Rep_run} \phi \quad 0 \quad K2)\)
  \quad \land \quad \forall n::nat. \quad \texttt{run_tick_count} \phi K2 (Suc n) \leq \texttt{run_tick_count} \phi K1 n\)
  \((\text{is} \quad (?P \equiv \neg \texttt{P'}))\)
\end{verbatim}

\begin{verbatim}
proof
\{ fix r::('a run)
  assume \((r \in ?P)\)
  \quad \text{thus} \((?P \subseteq ?P')\)
  \{ fix r::('a run)
    assume \((r \in ?P')\)
    \quad \text{hence} \((\forall n::nat. \quad \texttt{run_tick_count} r K2 n \leq \texttt{run_tick_count_strictly} r K1 n)\)
    \quad \text{by simp}
    \quad \text{hence} \(1::\forall n::nat. \quad \texttt{tick_count} r K2 n \leq \texttt{tick_count_strict} r K1 n)\)
    \quad \text{by simp}
    \quad \text{hence} \(\forall n::nat. \quad \texttt{tick_count_strict} r K2 (Suc n) \leq \texttt{tick_count_strict} r K1 n)\)
    \quad \text{by simp}
    \quad \text{hence} \(\forall n::nat. \quad \texttt{tick_count_strict} r K2 (Suc (Suc n)) \leq \texttt{tick_count_strict} r K1 (Suc (Suc n))\)
    \quad \text{by simp}
    \quad \text{hence} \(\forall n::nat. \quad \texttt{tick_count_strict} r K2 (Suc n) \leq \texttt{tick_count} r K1 n)\)
    \quad \text{by simp}
    \quad \text{ultimately have} \(\texttt{tick_count} r K2 0 = 0\)\)
    \quad \text{by simp}
    \quad \text{hence} \(\neg \text{hanlet} \quad ((\text{Rep_run} r \quad K2))\)
    \quad \text{unfolding tick_count_strict_def by simp}
    \quad \text{with} \quad \text{have} \((r \in ?P')\)
    \quad \text{by simp}
  \}\}
\end{verbatim}

Some properties of \texttt{run_tick_count}, \texttt{tick_count} and \texttt{Suc}:
Lemma \( \text{run\_tick\_count\_suc} \):
\[
\text{run\_tick\_count\ r\ c\ (Suc\ n)} = \begin{cases} 
\text{if}\ \text{hamlet}\ ((\text{Rep}\_\text{run}\ r)\ (\text{Suc}\ n)\ c) \\
\text{then}\ \text{Suc}\ (\text{run\_tick\_count}\ r\ c\ n) \\
\text{else}\ \text{run\_tick\_count}\ r\ c\ n.
\end{cases}
\]

By simp

Corollary \( \text{tick\_count\_suc} \):
\[
\text{tick\_count\ r\ c\ (Suc\ n)} = \begin{cases} 
\text{if}\ \text{hamlet}\ ((\text{Rep}\_\text{run}\ r)\ (\text{Suc}\ n)\ c) \\
\text{then}\ \text{Suc}\ (\text{tick\_count}\ r\ c\ n) \\
\text{else}\ \text{tick\_count}\ r\ c\ n.
\end{cases}
\]

By (simp add: tick_count_is_fun)

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

Lemma \( \text{card\_suc} \):
\[
\text{card}\ \{i. i \leq (\text{Suc}\ n) \land P\ i\} = \text{card}\ \{i. i \leq n \land P\ i\} + \text{card}\ \{i. i = (\text{Suc}\ n) \land P\ i\}
\]

Proof -

Have \( \{i. i \leq n \land P\ i\} \cap \{i. i = (\text{Suc}\ n) \land P\ i\} = \emptyset \) by auto

Moreover have \( \{i. i \leq n \land P\ i\} \cup \{i. i = (\text{Suc}\ n) \land P\ i\} = \{i. i \leq (\text{Suc}\ n) \land P\ i\} \) by auto

Moreover have \text{finite} \( \{i. i \leq n \land P\ i\} \) by simp

Moreover have \text{finite} \( \{i. i = (\text{Suc}\ n) \land P\ i\} \) by simp

Ultimately show \( \text{thesis} \) using \text{card\_Un\_disjoint} [of \( \{i. i \leq n \land P\ i\} \) \( \{i. i = (\text{Suc}\ n) \land P\ i\} \)] by simp

Qed

Lemma \( \text{card\_le\_leq} \):
\[
\text{card}\ \{i::\text{nat}. i < n \land P\ i\} = \text{card}\ \{i. i \leq m \land P\ i\} + \text{card}\ \{i. i = n \land P\ i\}
\]

Proof -

Have \( \{i::\text{nat}. i < n \land P\ i\} \cap \{i. i = n \land P\ i\} = \emptyset \) by auto

Moreover with assms have \( \{i::\text{nat}. i < n \land P\ i\} \cup \{i. i = n \land P\ i\} = \{i. i < n \land P\ i\} \) by auto

Moreover have \text{finite} \( \{i. n < i \land P\ i\} \) by simp

Moreover have \text{finite} \( \{i. i = n \land P\ i\} \) by simp

Ultimately show \( \text{thesis} \) using \text{card\_Un\_disjoint} [of \( \{i. n < i \land P\ i\} \) \( \{i. i = n \land P\ i\} \)] by simp

Qed

Lemma \( \text{card\_le\_leq\_0} \):
\[
\text{card}\ \{i::\text{nat}. i \leq n \land P\ i\} = \text{card}\ \{i. i < n \land P\ i\} + \text{card}\ \{i. i = n \land P\ i\}
\]

Proof -

Have \( \{i::\text{nat}. i < n \land P\ i\} \cap \{i. i = n \land P\ i\} = \emptyset \) by auto

Moreover have \( \{i. i < n \land P\ i\} \cup \{i. i = n \land P\ i\} = \{i. i \leq n \land P\ i\} \) by auto

Moreover have \text{finite} \( \{i. i < n \land P\ i\} \) by simp

Moreover have \text{finite} \( \{i. i = n \land P\ i\} \) by simp

Ultimately show \( \text{thesis} \) using \text{card\_Un\_disjoint} [of \( \{i. i < n \land P\ i\} \) \( \{i. i = n \land P\ i\} \)] by simp

Qed

Lemma \( \text{card\_mm} \):
\[
\text{card}\ \{i::\text{nat}. i < n \land P\ i\} = \text{card}\ \{i. i \leq m \land P\ i\} + \text{card}\ \{i. n < i \land i < n \land P\ i\}
\]

Proof -

Have \( 1::\{i::\text{nat}. i < n \land P\ i\} \cap \{i. n < i \land i < n \land P\ i\} = \emptyset \) by auto

From assms have \( \forall i::\text{nat}. i < n = (i \leq n) \lor (m < i \land i < n) \)
8.1. STUTTERING INVARIANCE

using less_trans by auto

hence 2:
⟨\{i::nat. i < n \land P i}\} = \{i. i < m \land i < n \land P i\}\rangle 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\}\rangle by simp

from card_Un_disjoint[OF 3 4 1] 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:\langle\{i::nat. i < m \land P i\} \cap \{i. m \leq i \land i < n \land P i\} = {}\rangle 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:\langle\{i::nat. i < n \land P i\} = \{i. i < m \land P i\} \cup \{i. m \leq i \land i < n \land P i\}\rangle by blast

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

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

from card_Un_disjoint[OF 3 4 1] 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

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)

case True

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

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

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\}\rangle 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 ((Rep_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: \langle f n < k \land k < f (Suc n) \land hamlet ((Rep_run r) k c)\rangle

}
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 \( \langle \; \text{dilating} \; f \; \text{sub} \; r \rangle \) shows \( \langle \; \text{tick_count} \; r \; c \; (f \; (Suc \; n)) = \text{tick_count} \; r \; c \; (f \; n) + \text{card} \; \{ k. \; k = f \; (Suc \; n) \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} \rangle \)
proof -
  have smf: \( \langle \; \text{strict_mono} \; f \rangle \) using assms dilating_def dilating_fun_def by blast
  moreover have \( \langle \; \text{finite} \; \{ k. \; k \leq f \; n \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} \rangle \) by simp
  moreover have \( \langle \; \text{finite} \; \{ k. \; f \; n < k \land k \leq f \; (Suc \; n) \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} \rangle \) by simp
  ultimately have \( \langle \; \{ k. \; k \leq f \; n \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} = \bigcup \{ k. \; f \; n < k \land k \leq f \; (Suc \; n) \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} \rangle \) by (simp add: nat_interval_union strict_mono_less_eq)
  moreover have \( \langle \; \text{finite} \; \{ k. \; f \; n < k \land k \leq f \; (Suc \; n) \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} \rangle \) by simp
  ultimately have \( \langle \; \text{card} \; \{ k. \; k \leq f \; (Suc \; n) \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} = \text{card} \; \{ k. \; k \leq f \; n \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} + \text{card} \; \{ k. \; f \; n < k \land k \leq f \; (Suc \; n) \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} \rangle \) by (simp add: * card_Un_disjoint)
  moreover from no_tick_before_suc[OF assms] have \( \langle \; \{ k. \; f \; n < k \land k \leq f \; (Suc \; n) \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} = \{ k. \; k = f \; (Suc \; n) \land \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \} \rangle \) using smf strict_mono_less by fastforce
  ultimately show \(?thesis \) by (simp add: tick_count_def)
qed

corollary tick_count_f_suc:
  assumes \( \langle \; \text{dilating} \; f \; \text{sub} \; r \rangle \) shows \( \langle \; \text{tick_count} \; r \; c \; (f \; (Suc \; n)) = \text{tick_count} \; r \; c \; (f \; n) + \text{if} \; \text{hamlet} \; ((\text{Rep_run} \; r) \; (f \; (Suc \; n)) \; c) \; \text{then} \; 1 \; \text{else} \; 0 \rangle \) using tick_count_fsuc[OF assms] card_sing_prop[of \( \langle \; f \; (Suc \; n) \; \lambda k. \; \text{hamlet} \; ((\text{Rep_run} \; r) \; k) \; c \rangle \)] by simp

corollary tick_count_f_suc_suc:
  assumes \( \langle \; \text{dilating} \; f \; \text{sub} \; r \rangle \) shows \( \langle \; \text{tick_count} \; r \; c \; (f \; (Suc \; n)) = \text{if} \; \text{hamlet} \; ((\text{Rep_run} \; r) \; (f \; (Suc \; n)) \; c) \; \text{then} \; \text{Suc} \; (\text{tick_count} \; r \; c \; (f \; n)) \; \text{else} \; \text{tick_count} \; r \; c \; (f \; n) \rangle \) using tick_count_f_suc[OF assms] by simp

lemma tick_count_f_suc_sub:
  assumes \( \langle \; \text{dilating} \; f \; \text{sub} \; r \rangle \) shows \( \langle \; \text{tick_count} \; r \; c \; (f \; (Suc \; n)) = \text{if} \; \text{hamlet} \; ((\text{Rep_run_sub} \; (f \; (Suc \; n)) \; c) \; \text{then} \; \text{Suc} \; (\text{tick_count} \; r \; c \; (f \; n)) \; \text{else} \; \text{tick_count} \; r \; c \; (f \; n) \rangle \) using tick_count_f_suc_suc[OF assms] by simp

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

lemma tick_count_latest:
  assumes (dilating f sub r)
  and \( f \in \text{range}(f) \leq n \land (\forall k. f(n) < k \land k \leq n \rightarrow (\exists k_0. f(k_0) = k)) \)
  shows \( \text{tick_count r c n = tick_count r c (f(n))} \)
proof -
  have union: \( (i. i \leq n \land \text{hamlet}(\text{Rep_run r} i c)) = \)
   \( (i. i \leq f(n) \land \text{hamlet}(\text{Rep_run r} i c)) \)
   \( \cup (i. f(n) < i \land i \leq n \land \text{hamlet}(\text{Rep_run r} i c)) \) using assms(2) by auto
  have partition: \( (i. i \leq f(n) \land \text{hamlet}(\text{Rep_run r} i c)) \)
   \( \cap (i. f(n) < i \land i \leq n \land \text{hamlet}(\text{Rep_run r} i c)) = \{\} \)
   by (simp add: disjoint_iff_not_equal)
  from assms have \( (i. f(n) < i \land i \leq n \land \text{hamlet}(\text{Rep_run r} i c)) = \{\} \)
   using no_tick_sub by fastforce
  with union and partition show \( ?\text{thesis} \) 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 \( \text{tick_count sub c n = tick_count r c (f(n))} \)
proof -
  have \( \text{tick_count sub c n = card}(i. i \leq n \land \text{hamlet}(\text{Rep_run sub} i c)) \)
  using tick_count_def[of sub c n].
  also have \( \ldots = \text{card}(\text{image} f(i. i \leq n \land \text{hamlet}(\text{Rep_run sub} i c))) \)
  using assms dilating_def dilating_injects[of assms] by (simp add: card_image)
  also have \( \ldots = \text{card}(i. i \leq f(n) \land \text{hamlet}(\text{Rep_run r} i c)) \)
  using dilated_prefix[of assms, symmetric, of f n c] by simp
  also have \( \ldots = \text{tick_count r c (f(n))} \)
  using tick_count_def[of r c f n] by simp
  finally show \( ?\text{thesis} \).
qed

corollary run_tick_count_sub:
  assumes (dilating f sub r)
  shows \( \text{run_tick_count sub c n = run_tick_count r c (f(n))} \)
proof -
  have \( \text{run_tick_count sub c n = tick_count sub c n} \)
  using tick_count_is_fun[of sub c n, symmetric].
  also from tick_count_sub[of assms] have \( \ldots = \text{tick_count r c (f(n))} \).
  also have \( \ldots = \#(r c (f(n))) \) using tick_count_is_fun[of r c f n] by simp
  finally show \( ?\text{thesis} \).
qed

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

lemma tick_count_strict_0:
  assumes (dilating f sub r)
  shows \( \text{tick_count_strict c (f(0)) = 0} \)
proof -
  from assms have \( (f(0) = 0) \) by (simp add: dilating_def dilating_fun_def)
  thus \( ?\text{thesis} \) 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 \( (f(n) < k \land k < (f(Suc n))) \)
  shows \( \text{tick_count_strict r c k = tick_count_strict r c (f(Suc n))} \)
proof -
  from assms(1) have smf: strict_mono f by (simp add: dilating_def dilating_fun_def)
  from assms(2) have f n < k by simp
  hence (∀i. k ≤ i → f n < i) by simp
  with no_tick_beforeSuc[OF assms(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 (image f {i. i < n ∧ hamlet ((Rep_run sub) i c)})⟩ using card_image
  also have ⟨... = card {i. i < f n ∧ hamlet ((Rep_run r) i c)}⟩ using dilated_strict_prefix[OF assms, symmetric, of ⟨n⟩ ⟨c⟩]
  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⟩
  shows ⟨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_strict_def[of ⟨sub⟩ ⟨c⟩ ⟨n⟩].
  also have ⟨... = card (image f {i. i < n ∧ hamlet ((Rep_run sub) i c)})⟩ using assms dilating_def dilating_injects[OF assms] by simp
  also have ⟨... = card {i. i < f n ∧ hamlet ((Rep_run r) i c)}⟩ using dilated_strict_prefix[OF assms, symmetric, of ⟨n⟩ ⟨c⟩] by simp
  finally show ?thesis .
qed

The tick count on any clock can only increase.

lemma mono_tick_count:
  mono (λk. 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⟩
  shows ⟨∃x0. f n0 = x ⟹ (∃n. f n < n ∧ (∀k. f n < k ∧ k ≤ n → (∀x0. f k0 = k)))⟩
proof (induction n)
  case 0
  with assms 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 ⟨∃x0. f n0 = x⟩)
    case True
8.1. STUTTERING INварIANCE

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

hence \((f(n) < (\text{Suc } n) \land (\forall k. f(n) < k \land k \leq (\text{Suc } n) \rightarrow (\exists k_0. f(k_0) = k)))\)

using Suc.prems Suc_leI le_antisym by blast

thus \(?thesis by blast

next

case False

from Suc.IH[OF this] obtain \( n_p \)

where \((f(n_p) < n \land (\forall k. f(n_p) < k \land k \leq n \rightarrow (\exists k_0. f(k_0) = k)))\) by blast

hence \((f(n_p < \text{Suc } n) \land (\forall k. f(n_p) < k \land k \leq n \rightarrow (\exists k_0. f(k_0) = k)))\) by simp

with Suc(2) have \((f(n_p) < (\text{Suc } n) \land (\forall k. f(n_p) < k \land k \leq (\text{Suc } n) \rightarrow (\exists k_0. f(k_0) = k)))\)

using le_Suc_eq by auto

thus \(?thesis by blast

qed

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

lemma strict_mono_suc:

assumes \((\text{strict_mono } f)\)

and \((f(n) < \text{Suc } (f(n))\)

shows \((\text{Suc } n)\)

proof

from assms(2) have \((f(sn) = \text{Suc } (f(n)))\) by simp

with strict_mono_less[OF assms(1)] have \((sn = \text{Suc } n)\) by simp

moreover have \((sn = \text{Suc } n)\)

proof

{ assume \((sn > \text{Suc } n)\)

  from this obtain \( i \) where \((n < i \land i < sn)\) by blast

  hence \((f(n) < f(i) \land f(i) < f(sn))\) using assms(1) by (simp add: strict_mono_def)

  with assms(2) have False by simp

} thus \(?thesis using not_less by blast

qed

ultimately show \(?thesis by (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 next_non_stuttering:

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

and \((f(n) < \text{Suc } (f(n)))\)

shows \((f(n) < \text{Suc } n)\)

proof

from assms(2) have \((f(sn) = \text{Suc } n)\)

with smf assms(3) have \((f(sn) = \text{Suc } n)\) using strict_mono_less by fastforce

have \((f(n) < f(n))\)

proof

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

  hence \((\exists k. f(n) < k < f(n))\) using \(\text{Suc_lessI}\) assms(3) by fastforce

  hence \((\exists k. f(n) < k < f(n))\) using \(\text{Suc_lessI}\) by blast

  with \(\text{Suc_lessI}\) have False by blast

} thus \(?thesis using not_less by blast

qed

hence \((\text{Suc } n)\) using assms(3) smf using strict_mono_less_eq by fastforce

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

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

**Lemma dil_tick_count:**

assumes \( \langle \text{sub} \ll r \rangle \) and \( \langle \forall n. \text{run_tick_count sub a n} \leq \text{run_tick_count sub b n} \rangle \)

shows \( \langle \text{run_tick_count r a n} \leq \text{run_tick_count r b n} \rangle \)

**Proof**

from assms(1) is_subrun_def obtain f where \( \ast : \text{dilating f sub r} \) by blast

show \( \ast : \text{dilating f sub r} \)

proof (induction n)

- from assms(2) have \( \langle \text{run_tick_count sub a 0} \leq \text{run_tick_count sub b 0} \rangle \) ..

- with \( \text{run_tick_count_sub[OF \ast, of _ 0]} \) have \( \langle \text{run_tick_count r a (f 0)} \leq \text{run_tick_count r b (f 0)} \rangle \) by simp

moreover from \( \ast \) have \( \langle f 0 = 0 \rangle \) by (simp add: dilating_def dilating_fun_def)

ultimately show \( \ast : \text{dilating f sub r} \)

next

- case \( \langle \exists n. f n = \text{Suc n'} \rangle \)

- case True

from this obtain \( n \) where \( fn0 : (f n = \text{Suc n'}) \) by blast

show \( \ast : \text{dilating f sub r} \)

proof (cases \( \langle \text{hamlet ((Rep_run sub) n a)} \rangle \))

- case True

have \( \langle \text{run_tick_count r a (f n)} \leq \text{run_tick_count r b (f n)} \rangle \) using assms(2) run_tick_count_sub[OF \ast] by simp

thus \( \ast : \text{dilating f sub r} \) by \( \langle \text{simp add: \ast : \text{dilating f sub r}} \rangle \)

next

- case False

hence \( \langle \neg \text{hamlet ((Rep_run r) \ (\text{Suc n'}) a)} \rangle \)

using \( \ast \) \( \langle \text{ticks_sub[OF Suc.IH no_tick_sub]} \rangle \) by fastforce

thus \( \ast : \text{dilating f sub r} \) by \( \langle \text{simp add: Suc.IH le_SucI}} \rangle \)

qed

next

- case False

thus \( \ast : \text{dilating f sub r} \) using \( \ast : \text{dilating f sub r} \)

qed

Time does not progress during stuttering instants.

**Lemma stutter_no_time:**

assumes \( \langle \text{dilating f sub r} \rangle \) and \( \langle \forall k. f n < k \land k \leq m \Rightarrow (\neg \exists k0. f k0 = k) \rangle \)

and \( \langle m > f n \rangle \)

shows \( \langle \text{time ((Rep_run r) m c)} = \text{time ((Rep_run r) (f n) c)} \rangle \)

**Proof**

from assms have \( \langle \forall k. k < m - (f n) \Rightarrow (\exists k0. f k0 = k) \rangle \) by simp

hence \( \langle \forall k. k < m - (f n) \Rightarrow \text{time ((Rep_run r) (f n) c)} \rangle \)

using assms(1) by \( \langle \text{simp add: \langle \text{dilating_def dilating_fun_def}} \rangle \)

hence \( \langle \forall k. k < m - (f n) \Rightarrow \text{time ((Rep_run r) (f n) c)} \rangle \)

using bounded_suc_ind[of \( (\neg \exists k. \text{time ((Rep_run r) k c)} \) \( (f n) \)] by blast

from assms(3) obtain \( m0 \) where \( m0 : \text{Suc m0} = m - (f n) \) using Suc_diff_Suc by blast

with \( \ast : \text{dilating f sub r} \)

hence \( \langle \forall k. k < n - (f n) \Rightarrow \text{time ((Rep_run r) (f n) c)} \rangle \) by auto

moreover from \( m0 \) have \( \langle \text{Suc ((f n) + m0)} = m \) by simp

ultimately show \( \ast : \text{dilating f sub r} \)

qed
8.1. STUTTERING INVARIANCE

Lemma time_stuttering:
assumes ⟨dilating f sub r⟩
and ⟨time ((Rep_run sub) n c) = τ⟩
and $\forall k. f n < k \land k \leq n \implies (\exists k_0. f k_0 = k) \rangle$
and ⟨m > f n⟩
shows ⟨time ((Rep_run r) m c) = τ⟩
proof
- from assms(3) have ⟨time ((Rep_run r) m c) = time ((Rep_run r) (f n) c)⟩
  using stutter_no_time[OF assms(1,3,4)] by blast
also from assms(1,2) have ⟨time ((Rep_run r) (f n) c) = τ⟩ 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 ⟨dilating f sub r⟩
shows ⟨first_time sub c n t = first_time r c (f n) t⟩
proof
assume ⟨first_time sub c n t⟩
with before_first_time[OF this]
have *: ⟨time ((Rep_run sub) n c) = t \land (\forall m < n. time((Rep_run sub) m c) < t)⟩
by (simp add: first_time_def)
moreover have \(\forall n c. time (Rep_run sub n c) = time (Rep_run r (f n) c)\)
  using assms(1)
by (simp add: dilating_def)
ultimately have **: ⟨time ((Rep_run r) (f n) c) = t \land (\forall m < n. time((Rep_run r) (f m) c) < t)⟩
  by simp
have \(\forall m < f n. time ((Rep_run r) m c) < t)\)
proof
- { fix m assume hyp: m < f n
  have ⟨time ((Rep_run r) n c) < t)⟩ by simp
proof (cases \(\exists m_0. f m_0 = n)\)
  case True
  from this obtain m_0 where mm0: ⟨m = f m_0⟩ by blast
  with hyp have m0n: ⟨m_0 < n⟩ using assms(1)
  by (simp add: dilating_def dilating_fun_def strict_mono_less)
hence ⟨time ((Rep_run sub) m_0 c) < t)⟩ using * by blast
thus ?thesis by (simp add: mm0 m0n **)
next
  case False
  hence ⟨\exists f_n. f_m < m \land (\forall k. f_m < k \land k \leq 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 \land (\forall k. f_m < k \land k \leq m \implies (\exists k_0. f k_0 = k))⟩
  by blast
hence \(\forall m < f n. time ((Rep_run r) m c) = time ((Rep_run sub) m_p c)\)
  using time_stuttering[OF assms] by blast
also from hyp mp have ⟨f m_p < f n) by linarith
hence \(\forall m_p < n) using assms
  by (simp add:dilating_def dilating_fun_def strict_mono_less)
hence \(\forall m < f n. time ((Rep_run r) m c) < t) using * by simp
finally show ?thesis by simp
qed
with ** show \(\forall m < f n. time ((Rep_run r) m c) < t)\)
next
assume \(\forall m < f n)\)
hence \(\forall m < f n. time ((Rep_run r) m c) < t)\)
with ** show \(\forall m < f n. time ((Rep_run r) m c) < t)\)

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.

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

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.
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 \texttt{dil_inverse}.
proof

from \textit{assms}(1) have \(*\)::\(\forall n. \text{finite}\ \{i. f i \leq n\}\)
using finite_less_ub by (simp add: dilating_def dilating_fun_def)
from \textit{assms}(1) have \(\emptyset = 0 = 0\) by (simp add: dilating_def dilating_fun_def)
hence \(\forall n. 0 \in \{i. f i \leq n\}\) by simp
hence \(*\*)::\(\forall n. \ (i. f i \leq n) \neq \\emptyset\) by simp

\{ fix \ n assume \(\exists k > g n. f k \leq n\) unfolding \(\text{dil_inverse}\_\text{def}\)
using \textit{Max_in}[OF \(*\) \(*\*)\] by blast

from this obtain \(k\) where \textit{kprop}\[OF assms(1)\] by blast

\{ assume \(\forall k \leq g n. f k \leq n\) unfolding \text{dil_inverse_def}
using Max_gr_iff[OF \(*\) \(*\*)\] by blast \}

thus \(\text{thesis}\) using not_less by blast
qed

from this obtain \(k\) where \(k \leq g n \land f k > n\) by blast
hence \(f (g n) \geq f k \land f k > n\) using \textit{assms}(1)

\{ assume \(\exists k \leq g n. f k \leq n\) unfolding \text{dil_inverse_def}
using Max_gr_iff[OF \(*\) \(*\*)\] by blast \}

thus \(\text{thesis}\) using not_less by blast
qed

end

8.1.5 Main Theorems

theory Stuttering
imports StutteringLemmas
begin

Using the lemmas of the previous section about the invariance by stuttering of various properties 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.

\textbf{lemma} sporadic_sub:

\begin{itemize}
  \item \textit{assms}(\(\subseteq\))
\end{itemize}
and \(\exists c \text{ sporadic} \ \tau \ \text{on} \ \ c'_{\text{TESL}}\)
shows \(\exists r \in \ c \text{ sporadic} \ \tau \ \text{on} \ \ c'_{\text{TESL}}\)

\textbf{proof}

from \textit{assms}(1) \textit{is_subrun_def} obtain \(f\)
where \textit{dilating f sub r} by blast
hence \(\forall n. \text{ time}\ ((\text{Rep_run sub}) n c) = \text{ time}\ ((\text{Rep_run r}) \ (f n) c) \land \text{ hamlet}\ ((\text{Rep_run sub}) n c) = \text{ hamlet}\ ((\text{Rep_run r}) \ (f n) c)\) by (simp add: dilating_def)
moreover from \textit{assms}(2) have
\(\exists c \in \ (r. \exists n. \text{ hamlet}\ ((\text{Rep_run r}) n c) \land \text{ time}\ ((\text{Rep_run r}) n c') \ = \ \tau)\) by simp
8.1. STUTTERING INVARIANCE

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

Implications are preserved in a dilated run.

**Theorem implies_sub:**

assumes \( (\text{sub} \ll r) \)

and \( (\text{sub} \in [ [ c_1 \implies c_2 ]_{\text{TESL}} ] \)

shows \( r \in [ [ c_1 \implies c_2 ]_{\text{TESL}} ] \)

**Proof**

- from assms(1) is_subrun_def obtain \( f \) where \( (\text{dilating } f \text{ sub } r) \) by blast
moreover from assms(2) have \( (\text{sub} \in \{ r. \forall n. \text{hamlet } ((\text{Rep\_run r}) n c_1) \longrightarrow \text{hamlet } ((\text{Rep\_run r}) n c_2) \}) \) by simp
hence \( (\forall n. \text{hamlet } ((\text{Rep\_run sub}) n c_1) \longrightarrow \text{hamlet } ((\text{Rep\_run sub}) n c_2)) \) by simp
ultimately have \( (\forall n. \text{hamlet } ((\text{Rep\_run r}) n c_1) \longrightarrow \text{hamlet } ((\text{Rep\_run r}) n c_2)) \)
using \( \text{ticks\_imp\_ticks\_subk} \) ticks_sub by blast
thus \( ?\text{thesis} \) by simp
qed

**Theorem implies_not_sub:**

assumes \( (\text{sub} \ll r) \)

and \( (\text{sub} \in [ [ c_1 \implies \neg c_2 ]_{\text{TESL}} ] \)

shows \( r \in [ [ c_1 \implies \neg c_2 ]_{\text{TESL}} ] \)

**Proof**

- from assms(1) is_subrun_def obtain \( f \) where \( (\text{dilating } f \text{ sub } r) \) by blast
moreover from assms(2) have \( (\text{sub} \in \{ r. \forall n. \text{run\_tick\_count } r c_2 n \leq \text{run\_tick\_count_strictly } r c_1 n \}) \) by simp
hence \( (\forall n. \text{run\_tick\_count_strictly } r c_2 n \leq \text{run\_tick\_count_strictly } r c_1 n) \) by simp
from \( \text{dil\_tick\_count}[\text{OF assms(1) this}] \)
have \( (\forall n. \text{run\_tick\_count_strictly } r c_2 n \leq \text{run\_tick\_count_strictly } r c_1 n) \) by simp
thus \( ?\text{thesis} \) by simp
qed

Precedence relations are preserved in a dilated run.

**Theorem weakly_precedes_sub:**

assumes \( (\text{sub} \ll r) \)

and \( (\text{sub} \in [ [ c_1 \text{ weakly precedes } c_2 ]_{\text{TESL}} ] \)

shows \( r \in [ [ c_1 \text{ weakly precedes } c_2 ]_{\text{TESL}} ] \)

**Proof**

- from assms(1) is_subrun_def obtain \( f \) where \( *: (\text{dilating } f \text{ sub } r) \) by blast
from assms(2) have \( (\text{sub} \in \{ r. \forall n. \text{run\_tick\_count } r c_2 n \leq \text{run\_tick\_count } r c_1 n \}) \) by simp
hence \( (\forall n. \text{run\_tick\_count } r c_2 n \leq \text{run\_tick\_count } r c_1 n) \) by simp
from \( \text{dil\_tick\_count}[\text{OF assms(1) this}] \)
have \( (\forall n. \text{run\_tick\_count_strictly } r c_2 n \leq \text{run\_tick\_count_strictly } r c_1 n) \) by simp
thus \( ?\text{thesis} \) by simp
qed

**Theorem strictly_precedes_sub:**

assumes \( (\text{sub} \ll r) \)

and \( (\text{sub} \in [ [ c_1 \text{ strictly precedes } c_2 ]_{\text{TESL}} ] \)

shows \( r \in [ [ c_1 \text{ strictly precedes } c_2 ]_{\text{TESL}} ] \)

**Proof**

- from assms(1) is_subrun_def obtain \( f \) where \( *: (\text{dilating } f \text{ sub } r) \) by blast
from assms(2) have \( (\text{sub} \in \{ r. \forall n::\text{nat}. \text{run\_tick\_count } r c_2 n \leq \text{run\_tick\_count_strictly } r c_1 n \}) \) by simp
by simp
with strictly_precedes_alt_def2[of c2 (c1)] have
  (sub ∈ { g. (¬ hamlet ((Rep_run g) 0 c2)) ∧ (∀ n::nat. (run_tick_count g c2 (Suc n)) ≤ (run_tick_count g c1 n)))}
by blast
hence ((¬ hamlet ((Rep_run sub) 0 c2)) ∧ (∀ n::nat. (run_tick_count sub c2 (Suc n)) ≤ (run_tick_count sub c1 n)))
by simp

hence
  1:(¬ hamlet ((Rep_run sub) 0 c2)) ∧ (∀ n::nat. (tick_count sub c2 (Suc n)) ≤ (tick_count sub c1 n))
by (simp add: tick_count_is_fun)
have (∀ n::nat. (tick_count r c2 (Suc n)) ≤ (tick_count r c1 n))
proof -
  fix n::nat
  have (tick_count r c2 (Suc n) ≤ tick_count r c1 n)
  proof (cases ∃ n0. f n0 = n)
    case True — n is in the image of f
    from this obtain n0 where fn:(f n0 = n) by blast
    show ?thesis
    proof (cases ∃ sn0. f sn0 = Suc n0)
      case True — Suc n is in the image of f
      from this obtain sn0 where fn:(f sn0 = Suc n0) by blast
      with fn strict_monoSuc * have (sn0 = Suc n0)
      using dilating_def dilating_Fun_def by blast
      with 1 have (tick_count sub c2 sn0 ≤ tick_count sub c1 n0) by simp
      thus ?thesis using fn fn_tick_count_sub[DF *] by simp
    next
case False — Suc n is not in the image of f
    hence (¬ hamlet ((Rep_run r) (Suc n) c2))
    using * by (simp add: dilating_def dilating_Fun_def)
    hence (tick_count r c2 (Suc n) = tick_count r c2 n)
    by (simp add: tick_countSuc)
    also have (... = tick_count sub c2 n0)
    using fn tick_count_sub[DF *] by simp
    finally have (tick_count r c2 (Suc n) = tick_count sub c2 (Suc n0))
    moreover have (tick_count sub c2 n0 ≤ tick_count sub c2 (Suc n0))
    by (simp add: tick_countSuc)
    ultimately have (tick_count r c2 (Suc n) ≤ tick_count sub c2 (Suc n0))
    moreover have (tick_count sub c2 (Suc n0) ≤ tick_count sub c1 n0) using 1 by simp
    ultimately have (tick_count r c2 (Suc n) ≤ tick_count sub c1 n0) by simp
    thus ?thesis using tick_count_sub[DF *] 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
    np_prop:(f n_p < n ∧ (∀ k. f n_p < k ∧ k ≤ n → (∃ k0. f k0 = k))) by blast
  from tick_count_latest[DF * this] have
    (tick_count r c1 n = tick_count r c1 (f n_p))
    hence a:(tick_count r c1 n = tick_count sub c1 n_p)
    using tick_count_sub[DF *] by simp
    have b: (tick_count sub c2 (Suc n_p) ≤ tick_count sub c1 n_p) using 1 by simp
    show ?thesis
    proof (cases ∃ sn0. f sn0 = Suc n)
      case True — Suc n is in the image of f
      from this obtain sn0 where fn:(f sn0 = Suc n) by blast
    qed
8.1. STUTTERING INVARIANCE

from next_non_stuttering[OF * np_prop this] have sn_prop:(sn0 = Suc n0) .
with b have \langle \text{tick_count sub} c2 sn0 \leq \text{tick_count sub} c1 n0 \rangle by simp
thus \langle \text{thesis} \rangle using \langle \text{tick_count_sub[OF *]} \rangle fsn a by auto
next

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

hence \langle \text{hamlet ((Rep_run r) (Suc n) c2)} \rangle using \ast by (simp add: dilating_def dilating_run_def)

hence \langle \text{tick_count r c2 (Suc n) = \text{tick_count r c2 n}} \rangle by (simp add: tick_count Suc)

also have \langle \ldots = \text{tick_count sub} c2 n0 \rangle using np_prop tick_count_sub[OF *]

by (simp add: tick_count Suc)

finally have \langle \text{tick_count r c2 (Suc n) = \text{tick_count sub} c2 n0} \rangle .

moreover have \langle \text{tick_count sub} c2 n0 \leq \text{tick_count sub} c2 (Suc n0) \rangle

by (simp add: tick_count Suc)

ultimately have \langle \text{tick_count r c2 (Suc n) \leq \text{tick_count sub} c2 (Suc n0)} \rangle by simp

moreover have \langle \text{thesis} \rangle using np_prop mono_tick_count using a by linarith

\text{qed}


Thus \langle \text{thesis} \rangle ..

\text{qed}

moreover from 1 have \langle \text{hamlet ((Rep_run r) 0 c2)} \rangle using \ast \text{empty_dilated_prefix ticks_sub by fastforce}

ultimately show \langle \text{thesis} \rangle by (simp add: tick_count_is_fun strictly_precedes_alt2)

\text{qed}

Time delayed relations are preserved in a dilated run.

definition T_ESL_interpretation_atomic.simps(5) [of using T_ESL]

theorem time_delayed_sub:

assumes \langle \text{sub} \ll r \rangle

and \langle \text{sub} \in \{ a \ \text{a time-delayed by} \ \delta \tau \ \text{on} \ ms \ \text{implies} \ b \ \} T_ESL \rangle

shows \langle \text{r} \in \{ a \ \text{a time-delayed by} \ \delta \tau \ \text{on} \ ms \ \text{implies} \ b \ \} T_ESL \rangle

proof -

from assms(1) is_subrun_def obtain f where \langle \ast:(\text{dilating f sub r}) \text{ by blast} \rangle

from assms(2) have \langle \forall n. \text{hamlet ((Rep_run sub) n a)} \rightarrow (\forall m \geq n. \text{first_time_sub ms m (time (Rep_run sub) n ms) +} \delta \tau) \rightarrow \text{hamlet ((Rep_run sub) m b)} \rangle

using T_ESL_interpretation_atomic.simps(5) [of \langle \text{a} \rangle \langle \delta \tau \rangle \langle \text{ms} \rangle \langle \text{b} \rangle ] \text{ by simp}

hence \langle \text{\ast:(\forall n0. \text{hamlet ((Rep_run r) (f n0) a)} \rightarrow (\forall m0 \geq n0. \text{first_time_r ms (f m0) (time (Rep_run r) (f m0) ms) +} \delta \tau) \rightarrow \text{hamlet ((Rep_run r) (f m0) b)} \rangle \rangle \rangle

using first_time_image[OF \ast] dilating_def \ast \text{ by fastforce}

hence \langle \forall n. \text{hamlet ((Rep_run r) n a)} \rightarrow (\forall m \geq n. \text{first_time_r ms m (time (Rep_run r) n ms) +} \delta \tau) \rightarrow \text{hamlet ((Rep_run r) m b)} \rangle

proof -

\{ \text{fix n assume asss:hamlet ((Rep_run r) n a)} \}

from ticks_image_sub[OF \ast asss] obtain n0 where mfn0: \langle n = f n0 \rangle \text{ by blast}

with \ast asss have ft0:

\langle \forall m0 \geq n0. \text{first_time_r ms (f m0) (time (Rep_run r) (f m0) ms) +} \delta \tau \rightarrow \text{hamlet ((Rep_run r) (f m0) b)} \rangle \text{ by fastblast}

have \langle \forall m \geq n. \text{first_time_r ms m (time (Rep_run r) n ms) +} \delta \tau \rightarrow \text{hamlet ((Rep_run r) m b)} \rangle

proof -

\{ \text{fix m assume hyp:} \langle m \geq n \rangle \}

have \langle \text{first_time_r ms m (time (Rep_run r n ms) +} \delta \tau \rightarrow \text{hamlet (Rep_run r m b)} \rangle

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 \( \ast \) by (simp add: dilating_def dilating_fun_def)
  ultimately show \( ?\text{thesis using } ft0 \text{ hyp } nfn0 \) by (simp add: strict_mono_less_eq)
next
  case False
  thesis
proof (cases \( m = 0 \))
  case True
  hence \( \exists pm. m = Suc pm \) by (simp add: not0_implies_Suc)
  from this obtain \( pm \) where \( m = Suc pm \) by blast
  hence \( \forall c \text{. time } (\text{Rep-run } r \text{ n c}) = time (\text{Rep-run r n c}) \) using \( \ast \) by simp
  moreover have \( \forall c. time (\text{Rep-run r n c}) = time (\text{Rep-run r n c}) \) by blast
  ultimately have \( \forall c. time (\text{Rep-run r n c}) = time (\text{Rep-run r n c}) \) by blast
  thus \( ?\text{thesis using } \ast \) by blast
next
  case False
  thesis

Time relations are preserved through dilation of a run.

lemma tagrel_sub':
  assumes \( \text{sub } \ll r \) and \( \text{sub } \in \{ \text{time-relation } [c_1, c_2] \in R \}_{\text{TESL}} \)
  shows \( \forall n. \text{time } ((\text{Rep-run } r \text{ n c}_1), \text{time } ((\text{Rep-run } r \text{ n c}_2))) \)
proof -
  from assms(1) is_subrun_def obtain \( f \) where \( \ast \):dilating f sub r by blast
  moreover from assms(2) TESL_interpretation_atomic.simps(2) have
  \( \text{sub } \in \{ r. \forall n. \text{time } ((\text{Rep-run } r \text{ n c}_1), \text{time } ((\text{Rep-run } r \text{ n c}_2))) \} \) by blast
  hence \( \forall n. \text{time } ((\text{Rep-run sub n c}_1), \text{time } ((\text{Rep-run sub n c}_2))) \) by simp
  show \( ?\text{thesis using } \ast \) by simp
proof (induction n)
  case 0
  from 1 have \( \forall n. \text{time } ((\text{Rep-run sub 0 c}_1), \text{time } ((\text{Rep-run sub 0 c}_2))) \) by simp
  moreover from \( \ast \):dilating_def dilating_fun_def by (simp add: dilating_def dilating_fun_def)
  ultimately show \( ?\text{case by simp} \)
next
  case Suc
  thus \( ?\text{thesis using } \ast \) by simp
proof (cases \( \exists n_0. f n_0 = Suc n \))
  case True
  with \( \ast \):dilating_def dilating_fun_def by (simp add: dilating_def dilating_fun_def)
8.1. STUTTERING INVARIANCE

thus \( ? \text{thesis using } \text{Suc.IH by simp} \)
next
case False
from this obtain \( n_0 \) where \( \text{n0prop}(f \ n_0 = \text{Suc n}) \) by blast
from 1 have \( \forall \text{(Rep_run sub) n c_1}, \text{time ((Rep_run sub) n c_2))} \) by simp
moreover from \( \text{n0prop} \) have \( \text{time ((Rep_run sub) n c_1) = time ((Rep_run r) (\text{Suc n}) c_1)} \)
by \( \text{(simp add: dilating_def)} \)
moreover from \( \text{n0prop} \) have \( \text{time ((Rep_run sub) n_0 c_2) = time ((Rep_run r) (\text{Suc n}) c_2)} \)
by \( \text{(simp add: dilating_def)} \)
ultimately show \( ? \text{thesis by simp} \)
qed
qed

corollary tagrel_sub:
assumes \( (\text{sub} \subseteq \text{r}) \)
and \( (\text{sub} \subseteq \text{R}) \)
shows \( (\text{time-relation} \ [c_1, c_2] \subseteq \text{R}) \)
using \( \text{tagrel_sub}[\text{OF assms}] \) unfolding \text{TESL_interpretation_atomic.simps(3)} by simp

Time relations are also preserved by contraction

lemma tagrel_sub_inv:
assumes \( (\text{sub} \subseteq \text{r}) \)
and \( (\text{r} \subseteq \text{R}) \)
shows \( (\text{time-relation} \ [c_1, c_2] \subseteq \text{R}) \)
proof -
from \( \text{assms(1) is_subrun_def} \) obtain \( f \) where \( \text{df:dilating f sub r} \) by blast
moreover from \( \text{assms(2) TESL_interpretation_atomic.simps(2)} \) have
\( (\forall n. \text{(Rep_run r) n c_1}, \text{time ((Rep_run r) n c_2)}) \) by blast
hence \( (\forall n. \text{R (time ((Rep_run r) n c_1), time ((Rep_run r) n c_2))}) \) by simp
hence \( (\forall n. (f \ n_0 = n) \rightarrow \text{R (time ((Rep_run r) n c_1), time ((Rep_run r) n c_2))}) \) by simp
hence \( (\forall n_0. \text{R (time ((Rep_run r) (f n_0) c_1), time ((Rep_run r) (f n_0) c_2))}) \) by blast
moreover from \( \text{dilating_def df} \) have
\( (\forall n. \text{R (time ((Rep_run sub) n c) = time ((Rep_run r) (f n) c))}) \) by blast
ultimately have \( (\forall n_0. \text{R (time ((Rep_run sub) n_0 c_1), time ((Rep_run sub) n_0 c_2))}) \) by auto
thus \( ? \text{thesis by simp} \)
qed

Kill relations are preserved in a dilated run.

theorem kill_sub:
assumes \( (\text{sub} \subseteq \text{r}) \)
and \( (\text{r} \subseteq \text{c_1 kills c_2}) \)
shows \( (\text{r} \subseteq \text{c_1 kills c_2}) \)
proof -
from \( \text{assms(1) is_subrun_def} \) obtain \( f \) where \( \text{df:dilating f sub r} \) by blast
from \( \text{assms(2) TESL_interpretation_atomic.simps(8)} \) have
\( (\forall n. \text{hamlet (Rep_run sub n c_1) \rightarrow (\forall m \geq n. ~ \text{hamlet (Rep_run sub n c_2)})}) \) by simp
hence \( 1: (\forall n. \text{hamlet (Rep_run r (f n) c_1) \rightarrow (\forall m \geq n. ~ \text{hamlet (Rep_run r (f n) c_2)})}) \)
using \( \text{ticks_sub[OF \_]} \) by simp
hence \( (\forall n. \text{hamlet (Rep_run r (f n) c_1) \rightarrow (\forall m \geq (f n). ~ \text{hamlet (Rep_run r m c_2)})}) \)
proof -
\{ fix \( n \) assume \( \text{hamlet (Rep_run r (f n) c_1)} \)
with 1 have \( 2: (\forall m \geq n. ~ \text{hamlet (Rep_run r (f m) c_2)}) \) by simp
have \( (\forall m \geq (f n). ~ \text{hamlet (Rep_run r m c_2)}) \)
proof -
\{ fix \( m \) assume \( \text{h:m \geq f n} \)
have \( (\neg \text{hamlet (Rep_run r m c_2)}) \) proof (cases \( \exists m_0. f m_0 = m \) )
\}
case True
  from this obtain m0 where fm0 : fm0 = m by blast
  hence (m0 \geq 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 ticks_image_sub[OF \(*) by blast
qed

Thus \(?thesis\) by simp

We can now prove that all atomic specification formulae are preserved by the dilation of runs.

**Lemma atomic_sub:**

assumes \(\langle \text{sub} \ll r \rangle\)
and \(\langle \text{sub} \in \ll [\varphi]_{\text{TESL}} \rangle\)

shows \(\langle r \in \ll [\varphi]_{\text{TESL}} \rangle\)

using assms(2) atomic_sub_lemmas[OF assms(1)] by (cases \varphi, simp_all)

Finally, any TESL specification is invariant by stuttering.

**Theorem TESL_stuttering_invariant:**

assumes \(\langle \text{sub} \ll r \rangle\)

shows \(\langle \text{sub} \in \ll [S]_{\text{TESL}} =\Rightarrow r \in \ll [S]_{\text{TESL}} \rangle\)

proof (induction S)
  case Nil
  thus \(?case\) by simp
next
case (Cons a s)
  from Cons.prems have sa: \(\langle \text{sub} \in \ll [a]_{\text{TESL}} \rangle\) and sb: \(\langle \text{sub} \in \ll [s]_{\text{TESL}} \rangle\)
  using TESL_interpretation_image by simp
  from Cons.IH[OF sb] have \(\langle r \in \ll [s]_{\text{TESL}} \rangle\).
  moreover from atomic_sub[OF assms(1) sa] have \(\langle r \in \ll [a]_{\text{TESL}} \rangle\).
  ultimately show \(?case\) using TESL_interpretation_image by simp
qed

end
8.1. STUTTERING INVARIANCE

\[ \text{definition morphism\_tagconst} : \langle (x::'τ \text{ tag}\_const) \otimes (f::('τ::\text{linorder} \Rightarrow 'τ)) \Rightarrow ('τ\text{\_cst} \circ f)(\text{the}\_\text{tag}\_\text{const} x) \rangle \]
end

Applying a TESL morphism to an atomic formula only changes the dates.

\[ \text{overloading morphism\_TESL\_atomic} \equiv \langle \text{morphism} : ('τ TESL\_atomic} \Rightarrow ('τ::\text{linorder} \Rightarrow 'τ) \Rightarrow ('τ TESL\_atomic) \rangle \]
begin
\[ \text{definition morphism\_TESL\_atomic} : \langle (\Ψ::'τ TESL\_atomic) \otimes (f::('τ::\text{linorder} \Rightarrow 'τ)) = (\text{case } \Ψ \text{ of} \rangle \]
| (C \text{ sporadic } t \text{ on } C') \Rightarrow (C \text{ sporadic } (t \otimes f) \text{ on } C')
| (C \text{ implies } C') \Rightarrow (C \text{ implies } C')
| (C \text{ implies not } C') \Rightarrow (C \text{ implies not } C')
| (C \text{ time-delayed by } t \text{ on } C' \text{ implies } C'') \Rightarrow (C \text{ time-delayed by } t \otimes f \text{ on } C' \text{ implies } C'')
| (C \text{ weakly precedes } C') \Rightarrow (C \text{ weakly precedes } C')
| (C \text{ strictly precedes } C') \Rightarrow (C \text{ strictly precedes } C')
| (C \text{ kills } C') \Rightarrow (C \text{ kills } C')) \rangle \]
end

Applying a TESL morphism to a formula amounts to apply it to each atomic formula.

\[ \text{overloading morphism\_TESL\_formula} \equiv \langle \text{morphism} : ('τ TESL\_formula} \Rightarrow ('τ::\text{linorder} \Rightarrow 'τ) \Rightarrow ('τ TESL\_formula) \rangle \]
begin
\[ \text{definition morphism\_TESL\_formula} : \langle ((\Ψ::'τ TESL\_formula) \otimes (f::('τ::\text{linorder} \Rightarrow 'τ)) = \text{map } (\lambda x. x \otimes f) \Ψ) \rangle \]
end

Applying a TESL morphism to a configuration amounts to apply it to the present and future formulae. The past (in the context \( \Gamma \)) is not changed.

\[ \text{overloading morphism\_TESL\_config} \equiv \langle \text{morphism} : ('τ::\text{linordered\_field}) \text{ config} \Rightarrow ('τ \Rightarrow 'τ) \Rightarrow ('τ config) \rangle \]
begin
\[ \text{fun morphism\_TESL\_config} \]
where \[ \langle ((\Gamma, n \vdash \Ψ \triangleright \Phi) :: ('τ::\text{linordered\_field}) \text{ config}) \otimes (f::('τ \Rightarrow 'τ)) = (\Gamma, n \vdash (\Ψ \otimes f) \triangleright (\Phi \otimes f)) \rangle \]
end

A TESL formula is called consistent if it possesses Kripke-models in its denotational interpretation.

\[ \text{definition consistent} :: (('τ::\text{linordered\_field}) TESL\_formula} \Rightarrow \text{bool} \]
where \[ (\text{consistent } \Ψ \equiv [[ \Psi ]]_{\text{TESL}} \neq \emptyset) \]
If we can derive a consistent finite context from a TESL formula, the formula is consistent.

\[ \text{theorem consistency\_finite} : \]
assumes start : \[ (([], 0 \vdash \Psi \triangleright []) \Rightarrow^* (\Gamma_1, n_1 \vdash [] \triangleright [])) \]
and init\_invariant : \[ (\text{consistent\_context } \Gamma_1) \]
shows (consistent \( \Psi \))
proof -
have \[ * : \exists n. (([], 0 \vdash \Psi \triangleright []) \Rightarrow^n (\Gamma_1, n_1 \vdash [] \triangleright [])) \]
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**

\[
\text{const_nontick_run} :: \langle \\langle \text{clock} \rightarrow 't \tag{const} \rangle \Rightarrow ('t :: \text{linordered_field}) \text{run} \rangle (\Box \_ 80)
\]

where \(\Box f \equiv \text{Abs_run}(\lambda n c. (\text{False}, f c))\)

Ensure a clock ticks in a run at a given instant.

**definition**

\[
\text{set_tick} :: \langle ('t :: \text{linordered_field}) \text{run} \Rightarrow \text{nat} \Rightarrow \text{clock} \Rightarrow ('t) \text{run} \rangle
\]

where \(\text{set_tick} r k c = \text{Abs_run}(\lambda n c. \text{if } k = n \text{ then } (\text{True}, \text{time(Rep_run r k c)}) \text{ else } \text{Rep_run r k c})\)

Ensure a clock does not tick in a run at a given instant.

**definition**

\[
\text{unset_tick} :: \langle ('t :: \text{linordered_field}) \text{run} \Rightarrow \text{nat} \Rightarrow \text{clock} \Rightarrow ('t) \text{run} \rangle
\]

where \(\text{unset_tick} r k c = \text{Abs_run}(\lambda n c. \text{if } k = n \text{ then } (\text{False}, \text{time(Rep_run r k c)}) \text{ else } \text{Rep_run r k c})\)

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**

\[
\text{patch} :: \langle ('t :: \text{linordered_field}) \text{run} \Rightarrow \text{nat} \Rightarrow ('t) \text{run} \Rightarrow ('t) \text{run} \rangle (\Box >_{\_ 80})
\]

where \(\text{r} >_{r'} \equiv \text{Abs_run}(\lambda n c. \text{if } n \leq k \text{ then } \text{Rep_run (r)} n c \text{ else } \text{Rep_run (r')} n c)\)

For some infinite cases, the idea for a proof scheme looks as follows: if we can derive from the initial configuration \(\Box, 0 \vdash \Psi \triangleright \Box\) a start-point of a lasso \(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1\), and if we can traverse the lasso one time \(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rightarrow^{++} \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2\) 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**

\[\text{consistency_coinduct} : \]

assumes start : \((\Box, 0 \vdash \Psi \triangleright \Box) \rightarrow^{**} (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1)\)

and loop : \((\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \rightarrow^{++} (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)\)

and init_invariant : \(\langle \text{consistent_context } \Gamma_1 \rangle\)

and post_invariant : \(\langle \text{consistent_context } \Gamma_2 \rangle\)

and retract_condition : \(\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle \otimes (f :: 't \Rightarrow 't) = (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1)\)

shows \(\langle \text{consistent (} \Psi :: ('t :: \text{linordered_field})\text{TESL_formula)}\rangle\)

oops

end
Bibliography
