

A Formal Development of a Polychronous Polytimed Coordination Language

Hai NGuyen Van

Frederic Boulanger

Burkhart Wolff

April 10, 2019

Contents

1	A Gentle Introduction to TESL	5
1.1	Context	5
1.2	The TESL Language	7
1.2.1	Instantaneous Causal Operators	7
1.2.2	Temporal Operators	8
1.2.3	Asynchronous Operators	8
2	The Core of the TESL Language: Syntax and Basics	11
2.1	Syntactic Representation	11
2.1.1	Basic elements of a specification	11
2.1.2	Operators for the TESL language	11
2.1.3	Field Structure of the Metric Time Space	12
2.2	Defining Runs	16
3	Denotational Semantics	19
3.1	Denotational interpretation for atomic TESL formulae	19
3.2	Denotational interpretation for TESL formulae	20
3.2.1	Image interpretation lemma	20
3.2.2	Expansion law	20
3.3	Equational laws for TESL formulae denotationally interpreted	21
3.4	Decreasing interpretation of TESL formulae	21
3.5	Some special cases	23
3.5.1	Symbolic Primitives for Runs	24
3.6	Semantics of Primitive Constraints	24
3.6.1	Defining a method for witness construction	25
3.7	Rules and properties of consistence	25
3.8	Major Theorems	26
3.8.1	Fixpoint lemma	26
3.8.2	Expansion law	26
3.9	Equational laws for TESL formulae denotationally interpreted	26
3.9.1	General laws	26
3.9.2	Decreasing interpretation of TESL formulae	27

4	Operational Semantics	31
4.1	Operational steps	31
4.2	Basic Lemmas	33
5	Equivalence of Operational and Denotational Semantics	37
5.1	Stepwise denotational interpretation of TESL atoms	37
5.2	Coinduction Unfolding Properties	40
5.3	Interpretation of configurations	44
6	Main Theorems	51
6.1	Initial configuration	51
6.2	Soundness	51
6.3	Completeness	55
6.4	Progress	57
6.5	Local termination	67
7	Properties of TESL	69
7.1	Stuttering Invariance	69
7.1.1	Definition of stuttering	69
7.1.2	Stuttering Lemmas	70
7.1.3	Lemmas used to prove the invariance by stuttering . .	71
7.1.4	Main Theorems	94

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 sub-models 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).

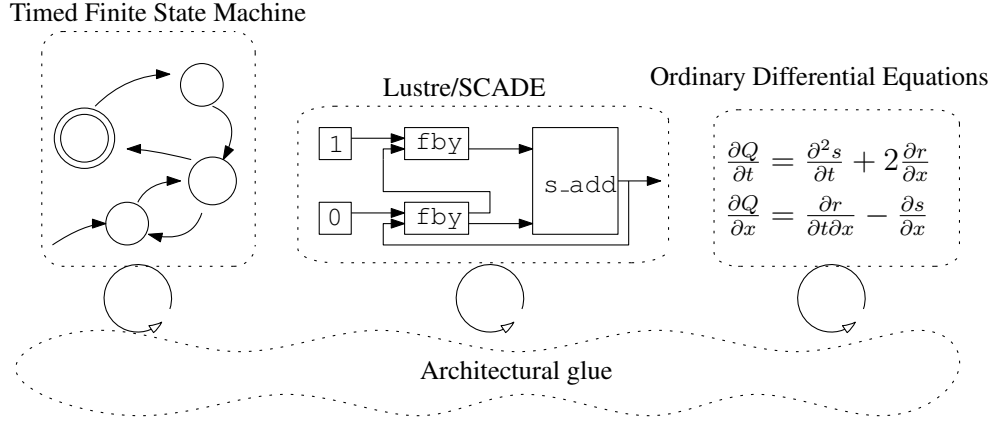


Figure 1.1: A Heterogeneous Timed System Model

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

The elements of Σ^∞ 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 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 times 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 will consider only linear relations here so that finding solutions 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 tell when ticks have to occur, then 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 instant or at the same instant. This can also be expressed by saying that at each instant, the number of ticks on *c2* since the beginning of the run must be lower or equal to the number of ticks on *c1*.
- **c1 strictly precedes c2** means that for each tick on *c2*, there must be at least one tick on *c1* at a previous instant. This can also be

¹See <http://wdi.supelec.fr/software/TESL/GalleryEngine> for more details

expressed by saying that at each instant, the number of ticks on *c2* from the beginning of the run to this instant must be lower or equal to the number of ticks on *c1* from the beginning of the run to the previous instant.

Chapter 2

The Core of the TESL Language: Syntax and Basics

```
theory TESL
imports Main
```

```
begin
```

2.1 Syntactic Representation

We define here the syntax of TESL specifications.

2.1.1 Basic elements of a specification

The following items appear in specifications:

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

```
datatype    clock          = Clk ⟨string⟩
type-synonym instant-index = ⟨nat⟩
```

```
datatype 'τ tag-const =
  TConst 'τ (τcst)
```

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 =
```

<i>SporadicOn</i>	$\langle \text{clock} \rangle \langle ' \tau \text{ tag-const} \rangle \langle \text{clock} \rangle$	$(- \text{ sporadic - on - } 55)$
<i>TagRelation</i>	$\langle \text{clock} \rangle \langle \text{clock} \rangle \langle (' \tau \text{ tag-const} \times ' \tau \text{ tag-const}) \Rightarrow \text{bool} \rangle$	$(\text{time-relation } [-, -] \in - 55)$
<i>Implies</i>	$\langle \text{clock} \rangle \langle \text{clock} \rangle$	$(\text{infixr implies } 55)$
<i>ImpliesNot</i>	$\langle \text{clock} \rangle \langle \text{clock} \rangle$	$(\text{infixr implies not } 55)$
<i>TimeDelayedBy</i>	$\langle \text{clock} \rangle \langle ' \tau \text{ tag-const} \rangle \langle \text{clock} \rangle \langle \text{clock} \rangle$	$(- \text{ time-delayed by - on - implies - } 55)$
<i>WeaklyPrecedes</i>	$\langle \text{clock} \rangle \langle \text{clock} \rangle$	$(\text{infixr weakly precedes } 55)$
<i>StrictlyPrecedes</i>	$\langle \text{clock} \rangle \langle \text{clock} \rangle$	$(\text{infixr strictly precedes } 55)$
<i>Kills</i>	$\langle \text{clock} \rangle \langle \text{clock} \rangle$	$(\text{infixr kills } 55)$

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

type-synonym $' \tau \text{ TESL-formula} = \langle ' \tau \text{ TESL-atomic list} \rangle$

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* :: $\langle ' \tau \text{ TESL-atomic} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{positive-atom } (- \text{ sporadic - on } -) = \text{True} \rangle$
 $\mid \langle \text{positive-atom } - = \text{False} \rangle$

The *NoSporadic* function removes sporadic constraints from a TESL formula.

abbreviation *NoSporadic* :: $\langle ' \tau \text{ TESL-formula} \Rightarrow ' \tau \text{ TESL-formula} \rangle$
where
 $\langle \text{NoSporadic } f \equiv (\text{List.filter } (\lambda f_{\text{atom}}. \text{case } f_{\text{atom}} \text{ of } - \text{ sporadic - on } - \Rightarrow \text{False} \mid - \Rightarrow \text{True}) f) \rangle$

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* :: $(\text{field})\text{field}$

begin

fun *inverse-tag-const*

where $\langle \text{inverse } (\tau_{\text{cst}} t) = \tau_{\text{cst}} (\text{inverse } t) \rangle$

fun *divide-tag-const*

where $\langle \text{divide } (\tau_{\text{cst}} t_1) (\tau_{\text{cst}} t_2) = \tau_{\text{cst}} (\text{divide } t_1 t_2) \rangle$

fun *uminus-tag-const*

where $\langle \text{uminus } (\tau_{\text{cst}} t) = \tau_{\text{cst}} (\text{uminus } t) \rangle$

fun *minus-tag-const*

where $\langle \text{minus } (\tau_{\text{cst}} t_1) (\tau_{\text{cst}} t_2) = \tau_{\text{cst}} (\text{minus } t_1 t_2) \rangle$

definition $\langle one\text{-}tag\text{-}const \equiv \tau_{cst} 1 \rangle$

fun *times-tag-const*
where $\langle times (\tau_{cst} t_1) (\tau_{cst} t_2) = \tau_{cst} (times t_1 t_2) \rangle$

definition $\langle zero\text{-}tag\text{-}const \equiv \tau_{cst} 0 \rangle$

fun *plus-tag-const*
where $\langle plus (\tau_{cst} t_1) (\tau_{cst} t_2) = \tau_{cst} (plus t_1 t_2) \rangle$

instance proof

Multiplication is associative.

fix $a::\langle \tau::field\ tag\text{-}const \rangle$ **and** $b::\langle \tau::field\ tag\text{-}const \rangle$ **and** $c::\langle \tau::field\ tag\text{-}const \rangle$
obtain $u\ v\ w$ **where** $\langle a = \tau_{cst} u \rangle$ **and** $\langle b = \tau_{cst} v \rangle$ **and** $\langle c = \tau_{cst} w \rangle$
using *tag-const.exhaust* **by** *metis*
thus $\langle a * b * c = a * (b * c) \rangle$
by (*simp add: TESL.times-tag-const.simps*)
next

Multiplication is commutative.

fix $a::\langle \tau::field\ tag\text{-}const \rangle$ **and** $b::\langle \tau::field\ tag\text{-}const \rangle$
obtain $u\ v$ **where** $\langle a = \tau_{cst} u \rangle$ **and** $\langle b = \tau_{cst} v \rangle$ **using** *tag-const.exhaust* **by** *metis*
thus $\langle a * b = b * a \rangle$
by (*simp add: TESL.times-tag-const.simps*)
next

One is neutral for multiplication.

fix $a::\langle \tau::field\ tag\text{-}const \rangle$
obtain u **where** $\langle a = \tau_{cst} u \rangle$ **using** *tag-const.exhaust* **by** *blast*
thus $\langle 1 * a = a \rangle$
by (*simp add: TESL.times-tag-const.simps one-tag-const-def*)
next

Addition is associative.

fix $a::\langle \tau::field\ tag\text{-}const \rangle$ **and** $b::\langle \tau::field\ tag\text{-}const \rangle$ **and** $c::\langle \tau::field\ tag\text{-}const \rangle$
obtain $u\ v\ w$ **where** $\langle a = \tau_{cst} u \rangle$ **and** $\langle b = \tau_{cst} v \rangle$ **and** $\langle c = \tau_{cst} w \rangle$
using *tag-const.exhaust* **by** *metis*
thus $\langle a + b + c = a + (b + c) \rangle$
by (*simp add: TESL.plus-tag-const.simps*)
next

Addition is commutative.

fix $a::\langle \tau::field\ tag\text{-}const \rangle$ **and** $b::\langle \tau::field\ tag\text{-}const \rangle$
obtain $u\ v$ **where** $\langle a = \tau_{cst} u \rangle$ **and** $\langle b = \tau_{cst} v \rangle$ **using** *tag-const.exhaust* **by** *metis*

```

thus  $\langle a + b = b + a \rangle$ 
  by (simp add: TESL.plus-tag-const.simps)
next

```

Zero is neutral for addition.

```

fix  $a::\langle\tau::\text{field tag-const}\rangle$ 
obtain  $u$  where  $\langle a = \tau_{cst} u \rangle$  using tag-const.exhaust by blast
thus  $\langle 0 + a = a \rangle$ 
  by (simp add: TESL.plus-tag-const.simps zero-tag-const-def)
next

```

The sum of an element and its opposite is zero.

```

fix  $a::\langle\tau::\text{field tag-const}\rangle$ 
obtain  $u$  where  $\langle a = \tau_{cst} u \rangle$  using tag-const.exhaust by blast
thus  $\langle -a + a = 0 \rangle$ 
  by (simp add: TESL.plus-tag-const.simps
      TESL.uminus-tag-const.simps
      zero-tag-const-def)
next

```

Subtraction is adding the opposite.

```

fix  $a::\langle\tau::\text{field tag-const}\rangle$  and  $b::\langle\tau::\text{field tag-const}\rangle$ 
obtain  $u v$  where  $\langle a = \tau_{cst} u \rangle$  and  $\langle b = \tau_{cst} v \rangle$  using tag-const.exhaust by
metis
thus  $\langle a - b = a + -b \rangle$ 
  by (simp add: TESL.minus-tag-const.simps TESL.plus-tag-const.simps TESL.uminus-tag-const.simps)
next

```

Distributive property of multiplication over addition.

```

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  $\langle a = \tau_{cst} u \rangle$  and  $\langle b = \tau_{cst} v \rangle$  and  $\langle c = \tau_{cst} w \rangle$ 
  using tag-const.exhaust by metis
thus  $\langle (a + b) * c = a * c + b * c \rangle$ 
  by (simp add: TESL.plus-tag-const.simps
      TESL.times-tag-const.simps
      ring-class.ring-distrib(2))
next

```

The neutral elements are distinct.

```

show  $\langle (0::\langle\tau::\text{field tag-const}\rangle) \neq 1 \rangle$ 
  by (simp add: one-tag-const-def zero-tag-const-def)
next

```

The product of an element and its inverse is 1.

```

fix  $a::\langle\tau::\text{field tag-const}\rangle$  assume  $h::\langle a \neq 0 \rangle$ 
obtain  $u$  where  $\langle a = \tau_{cst} u \rangle$  using tag-const.exhaust by blast
moreover with  $h$  have  $\langle u \neq 0 \rangle$  by (simp add: zero-tag-const-def)
ultimately show  $\langle \text{inverse } a * a = 1 \rangle$ 

```

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 $a::\langle\tau::\text{field tag-const}\rangle$ **and** $b::\langle\tau::\text{field tag-const}\rangle$
obtain $u\ v$ **where** $\langle a = \tau_{cst}\ u \rangle$ **and** $\langle b = \tau_{cst}\ v \rangle$ **using** *tag-const.exhaust* **by**
metis
thus $\langle a\ \text{div}\ b = a * \text{inverse}\ b \rangle$
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 $\langle \text{inverse}\ (0::\langle\tau::\text{field tag-const}\rangle) = 0 \rangle$
by (*simp add: TESL.inverse-tag-const.simps zero-tag-const-def*)
qed
end

For comparing dates on clocks, we need an order on tags.

instantiation *tag-const* :: (*order*)*order*

begin

inductive *less-eq-tag-const* :: $\langle 'a\ \text{tag-const} \Rightarrow 'a\ \text{tag-const} \Rightarrow \text{bool} \rangle$
where
Int-less-eq[simp]: $\langle n \leq m \Longrightarrow (TConst\ n) \leq (TConst\ m) \rangle$

definition *less-tag*: $\langle (x::'a\ \text{tag-const}) < y \longleftrightarrow (x \leq y) \wedge (x \neq y) \rangle$

instance proof

show $\langle \bigwedge x\ y :: 'a\ \text{tag-const}. (x < y) = (x \leq y \wedge \neg y \leq x) \rangle$
using *less-eq-tag-const.simps less-tag* **by** *auto*

next

fix $x::'a\ \text{tag-const}$

from *tag-const.exhaust* **obtain** $x_0::'a$ **where** $\langle x = TConst\ x_0 \rangle$ **by** *blast*

with *Int-less-eq* **show** $\langle x \leq x \rangle$ **by** *simp*

next

show $\langle \bigwedge x\ y\ z :: 'a\ \text{tag-const}. x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z \rangle$
using *less-eq-tag-const.simps* **by** *auto*

next

show $\langle \bigwedge x\ y :: 'a\ \text{tag-const}. x \leq y \Longrightarrow y \leq x \Longrightarrow x = y \rangle$
using *less-eq-tag-const.simps* **by** *auto*

qed

end

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

```

instantiation tag-const :: (linorder)linorder
begin
  instance proof
    fix x::'a tag-const and y::'a tag-const
    from tag-const.exhaust obtain x0::'a where  $\langle x = TConst\ x_0 \rangle$  by blast
    moreover from tag-const.exhaust obtain y0::'a where  $\langle y = TConst\ y_0 \rangle$  by
    blast
    ultimately show  $\langle x \leq y \vee y \leq x \rangle$  using less-eq-tag-const.simps by fastforce
    qed
  end
end

```

2.2 Defining Runs

```

theory Run
imports TESL

```

```

begin

```

Runs are sequences of instants, each instant mapping a clock to a pair that whether the clock ticks or not and what 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  $\langle hamlet \equiv fst \rangle$ 
abbreviation time where  $\langle time \equiv snd \rangle$ 

```

```

type-synonym ' $\tau$  instant =  $\langle clock \Rightarrow (bool \times '\tau\ tag-const) \rangle$ 

```

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) ' $\tau$ ::linordered-field run =
   $\langle \{ \varrho :: nat \Rightarrow '\tau\ instant. \forall c. mono\ (\lambda n. time\ (\varrho\ n\ c)) \} \rangle$ 
proof
  show  $\langle (\lambda - . (True, \tau_{cst}\ 0)) \in \{ \varrho. \forall c. mono\ (\lambda n. time\ (\varrho\ n\ c)) \} \rangle$ 
  unfolding mono-def by blast
qed

```

```

lemma Abs-run-inverse-rewrite:

```

```

   $\langle \forall c. mono\ (\lambda n. time\ (\varrho\ n\ c)) \rangle \Longrightarrow Rep-run\ (Abs-run\ \varrho) = \varrho$ 
  by (simp add: Abs-run-inverse)

```

run-tick-count $\varrho\ K\ n$ counts the number of ticks on clock K in the interval $[0, n]$ of run ϱ .

```

fun run-tick-count ::  $\langle (' \tau :: linordered-field)\ run \Rightarrow clock \Rightarrow nat \Rightarrow nat \rangle$  ( $\#_{\leq} - -$ )
where

```


$$\begin{aligned}
\langle \#_{\leq} \varrho K 0 \rangle &= (\text{if hamlet } ((\text{Rep-run } \varrho) 0 K) \\
&\quad \text{then } 1 \\
&\quad \text{else } 0) \\
| \langle \#_{\leq} \varrho K (\text{Suc } n) \rangle &= (\text{if hamlet } ((\text{Rep-run } \varrho) (\text{Suc } n) K) \\
&\quad \text{then } 1 + (\#_{\leq} \varrho K n) \\
&\quad \text{else } (\#_{\leq} \varrho K n))
\end{aligned}$$

run-tick-count-strictly $\varrho K n$ counts the number of ticks on clock K in the interval $[0, n[$ of run ϱ .

fun *run-tick-count-strictly* :: $\langle 'a::\text{linordered-field} \rangle \text{ run} \Rightarrow \text{clock} \Rightarrow \text{nat} \Rightarrow \text{nat} \rangle (\#_{<}$
 $\text{--})$

where

$$\begin{aligned}
\langle \#_{<} \varrho K 0 \rangle &= 0 \\
| \langle \#_{<} \varrho K (\text{Suc } n) \rangle &= \#_{\leq} \varrho K n
\end{aligned}$$

definition *first-time* :: $\langle 'a::\text{linordered-field} \rangle \text{ run} \Rightarrow \text{clock} \Rightarrow \text{nat} \Rightarrow 'a \text{ tag-const} \Rightarrow \text{bool} \rangle$

where

$$\langle \text{first-time } \varrho K n \tau \equiv (\text{time } ((\text{Rep-run } \varrho) n K) = \tau) \wedge (\nexists n'. n' < n \wedge \text{time } ((\text{Rep-run } \varrho) n' K) = \tau) \rangle$$

lemma *before-first-time*:

assumes $\langle \text{first-time } \varrho K n \tau \rangle$

and $\langle m < n \rangle$

shows $\langle \text{time } ((\text{Rep-run } \varrho) m K) < \tau \rangle$

proof –

have $\langle \text{mono } (\lambda n. \text{time } ((\text{Rep-run } \varrho) n K)) \rangle$ **using** *Rep-run* **by** *blast*

moreover from *assms*(2) **have** $\langle m \leq n \rangle$ **using** *less-imp-le* **by** *simp*

moreover have $\langle \text{mono } (\lambda n. \text{time } ((\text{Rep-run } \varrho) n K)) \rangle$ **using** *Rep-run* **by** *blast*

ultimately have $\langle \text{time } ((\text{Rep-run } \varrho) m K) \leq \text{time } ((\text{Rep-run } \varrho) n K) \rangle$ **by** (*simp add:mono-def*)

moreover from *assms*(1) **have** $\langle \text{time } ((\text{Rep-run } \varrho) n K) = \tau \rangle$ **using** *first-time-def* **by** *blast*

moreover from *assms* **have** $\langle \text{time } ((\text{Rep-run } \varrho) m K) \neq \tau \rangle$ **using** *first-time-def* **by** *blast*

ultimately show *?thesis* **by** *simp*

qed

lemma *alt-first-time-def*:

assumes $\langle \forall m < n. \text{time } ((\text{Rep-run } \varrho) m K) < \tau \rangle$

and $\langle \text{time } ((\text{Rep-run } \varrho) n K) = \tau \rangle$

shows $\langle \text{first-time } \varrho K n \tau \rangle$

proof –

from *assms*(1) **have** $\langle \forall m < n. \text{time } ((\text{Rep-run } \varrho) m K) \neq \tau \rangle$ **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
```

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 ⟧TESL =
    { ρ. ∃ n::nat. hamlet ((Rep-run ρ) n K1) ∧ time ((Rep-run ρ) n K2) = τ }⟩
  | ⟨⟦ time-relation [K1, K2] ∈ R ⟧TESL =
    { ρ. ∀ n::nat. R (time ((Rep-run ρ) n K1), time ((Rep-run ρ) n K2)) }⟩
  | ⟨⟦ master implies slave ⟧TESL =
    { ρ. ∀ n::nat. hamlet ((Rep-run ρ) n master) ⟶ hamlet ((Rep-run ρ) n
slave) }⟩
  | ⟨⟦ master implies not slave ⟧TESL =
    { ρ. ∀ n::nat. hamlet ((Rep-run ρ) n master) ⟶ ¬ hamlet ((Rep-run ρ) n
slave) }⟩
  | ⟨⟦ master time-delayed by δτ on measuring implies slave ⟧TESL =
    — When master ticks, let's call @termt0 the current date on measuring. Then,
    at the first instant when the date on measuring is @termt0+δt, slave has to tick.
    { ρ. ∀ n. hamlet ((Rep-run ρ) n master) ⟶
      (let measured-time = time ((Rep-run ρ) n measuring) in
        ∀ m ≥ n. first-time ρ measuring m (measured-time + δτ)
        ⟶ hamlet ((Rep-run ρ) m slave)
      )
    }⟩
  | ⟨⟦ K1 weakly precedes K2 ⟧TESL =
```

$$\begin{aligned}
& \{ \varrho. \forall n::nat. (run\text{-}tick\text{-}count\ \varrho\ K_2\ n) \leq (run\text{-}tick\text{-}count\ \varrho\ K_1\ n) \} \\
& | \langle \llbracket K_1\ strictly\ precedes\ K_2 \rrbracket_{TESL} = \\
& \quad \{ \varrho. \forall n::nat. (run\text{-}tick\text{-}count\ \varrho\ K_2\ n) \leq (run\text{-}tick\text{-}count\text{-}strictly\ \varrho\ K_1\ n) \} \rangle \\
& | \langle \llbracket K_1\ kills\ K_2 \rrbracket_{TESL} = \\
& \quad \{ \varrho. \forall n::nat. hamlet\ ((Rep\text{-}run\ \varrho)\ n\ K_1) \longrightarrow (\forall m \geq n. \neg hamlet\ ((Rep\text{-}run\ \varrho)\ m\ K_2)) \} \rangle
\end{aligned}$$

3.2 Denotational interpretation for TESL formulae

fun *TESL-interpretation* :: $\langle (' \tau :: linordered\text{-}field) \text{ TESL}\text{-}formula \Rightarrow ' \tau \text{ run set} \rangle$ ($\llbracket - \rrbracket_{TESL}$) **where**

$$\begin{aligned}
& \langle \llbracket [] \rrbracket_{TESL} = \{ \cdot. True \} \rangle \\
& | \langle \llbracket \varphi \# \Phi \rrbracket_{TESL} = \llbracket \varphi \rrbracket_{TESL} \cap \llbracket \Phi \rrbracket_{TESL} \rangle
\end{aligned}$$

lemma *TESL-interpretation-homo*:

$$\langle \llbracket \varphi \rrbracket_{TESL} \cap \llbracket \Phi \rrbracket_{TESL} = \llbracket \varphi \# \Phi \rrbracket_{TESL} \rangle$$

by *auto*

3.2.1 Image interpretation lemma

theorem *TESL-interpretation-image*:

$$\langle \llbracket \Phi \rrbracket_{TESL} = \bigcap ((\lambda \varphi. \llbracket \varphi \rrbracket_{TESL}) \text{ ' set } \Phi) \rangle$$

proof (*induct* Φ)

case *Nil*

then show *?case* **by** *simp*

next

case (*Cons* *a* Φ)

then show *?case* **by** *auto*

qed

3.2.2 Expansion law

Similar to the expansion laws of lattices

theorem *TESL-interp-homo-append*:

$$\langle \llbracket \Phi_1 @ \Phi_2 \rrbracket_{TESL} = \llbracket \Phi_1 \rrbracket_{TESL} \cap \llbracket \Phi_2 \rrbracket_{TESL} \rangle$$

proof (*induct* Φ_1)

case *Nil*

then show *?case* **by** *simp*

next

case (*Cons* *a* Φ_1)

then show *?case* **by** *auto*

qed

3.3 Equational laws for TESL formulae denotationally interpreted

lemma *TESL-interp-assoc*:

$$\langle \llbracket (\Phi_1 @ \Phi_2) @ \Phi_3 \rrbracket_{TESL} = \llbracket \Phi_1 @ (\Phi_2 @ \Phi_3) \rrbracket_{TESL} \rangle$$

by *auto*

lemma *TESL-interp-commute*:

$$\text{shows } \langle \llbracket \Phi_1 @ \Phi_2 \rrbracket_{TESL} = \llbracket \Phi_2 @ \Phi_1 \rrbracket_{TESL} \rangle$$

by (*simp add: TESL-interp-homo-append inf-sup-aci(1)*)

lemma *TESL-interp-left-commute*:

$$\langle \llbracket \Phi_1 @ (\Phi_2 @ \Phi_3) \rrbracket_{TESL} = \llbracket \Phi_2 @ (\Phi_1 @ \Phi_3) \rrbracket_{TESL} \rangle$$

unfolding *TESL-interp-homo-append* **by** *auto*

lemma *TESL-interp-idem*:

$$\langle \llbracket \Phi @ \Phi \rrbracket_{TESL} = \llbracket \Phi \rrbracket_{TESL} \rangle$$

using *TESL-interp-homo-append* **by** *auto*

lemma *TESL-interp-left-idem*:

$$\langle \llbracket \Phi_1 @ (\Phi_1 @ \Phi_2) \rrbracket_{TESL} = \llbracket \Phi_1 @ \Phi_2 \rrbracket_{TESL} \rangle$$

using *TESL-interp-homo-append* **by** *auto*

lemma *TESL-interp-right-idem*:

$$\langle \llbracket (\Phi_1 @ \Phi_2) @ \Phi_2 \rrbracket_{TESL} = \llbracket \Phi_1 @ \Phi_2 \rrbracket_{TESL} \rangle$$

unfolding *TESL-interp-homo-append* **by** *auto*

lemmas *TESL-interp-aci = TESL-interp-commute TESL-interp-assoc TESL-interp-left-commute TESL-interp-left-idem*

lemma *TESL-interp-neutral1*:

$$\langle \llbracket [] @ \Phi \rrbracket_{TESL} = \llbracket \Phi \rrbracket_{TESL} \rangle$$

by *simp*

lemma *TESL-interp-neutral2*:

$$\langle \llbracket \Phi @ [] \rrbracket_{TESL} = \llbracket \Phi \rrbracket_{TESL} \rangle$$

by *simp*

3.4 Decreasing interpretation of TESL formulae

lemma *TESL-sem-decreases-head*:

$$\langle \llbracket \Phi \rrbracket_{TESL} \supseteq \llbracket \varphi \# \Phi \rrbracket_{TESL} \rangle$$

by *simp*

lemma *TESL-sem-decreases-tail*:

$$\langle \llbracket \Phi \rrbracket_{TESL} \supseteq \llbracket \Phi @ [\varphi] \rrbracket_{TESL} \rangle$$

by (*simp add: TESL-interp-homo-append*)

lemma $\langle \varphi \# \Phi = [\varphi] @ \Phi \rangle$ **by** *simp*

lemma *TESL-interp-formula-stuttering*:

assumes $\langle \varphi \in \text{set } \Phi \rangle$

shows $\langle \llbracket \varphi \# \Phi \rrbracket_{TESL} = \llbracket \Phi \rrbracket_{TESL} \rangle$

proof –

have $\langle \varphi \# \Phi = [\varphi] @ \Phi \rangle$ **by** *simp*

hence $\langle \llbracket \varphi \# \Phi \rrbracket_{TESL} = \llbracket [\varphi] \rrbracket_{TESL} \cap \llbracket \Phi \rrbracket_{TESL} \rangle$ **using** *TESL-interp-homo-append*
by *simp*

thus *?thesis* **using** *assms TESL-interpretation-image* **by** *fastforce*

qed

lemma *TESL-interp-decreases*:

$\langle \llbracket \Phi \rrbracket_{TESL} \supseteq \llbracket \varphi \# \Phi \rrbracket_{TESL} \rangle$

by (rule *TESL-sem-decreases-head*)

lemma *TESL-interp-remdups-absorb*:

$\langle \llbracket \Phi \rrbracket_{TESL} = \llbracket \text{remdups } \Phi \rrbracket_{TESL} \rangle$

proof (*induct* Φ)

case *Nil*

then show *?case* **by** *simp*

next

case (*Cons* *a* Φ)

then show *?case*

using *TESL-interp-formula-stuttering* **by** *auto*

qed

lemma *TESL-interp-set-lifting*:

assumes $\langle \text{set } \Phi = \text{set } \Phi' \rangle$

shows $\langle \llbracket \Phi \rrbracket_{TESL} = \llbracket \Phi' \rrbracket_{TESL} \rangle$

proof –

have $\langle \text{set } (\text{remdups } \Phi) = \text{set } (\text{remdups } \Phi') \rangle$

by (*simp add: assms*)

moreover have *fxpnt* Φ : $\langle \bigcap ((\lambda \varphi. \llbracket \varphi \rrbracket_{TESL}) ' \text{set } \Phi) = \llbracket \Phi \rrbracket_{TESL} \rangle$

by (*simp add: TESL-interpretation-image*)

moreover have *fxpnt* Φ' : $\langle \bigcap ((\lambda \varphi. \llbracket \varphi \rrbracket_{TESL}) ' \text{set } \Phi') = \llbracket \Phi' \rrbracket_{TESL} \rangle$

by (*simp add: TESL-interpretation-image*)

moreover have $\langle \bigcap ((\lambda \varphi. \llbracket \varphi \rrbracket_{TESL}) ' \text{set } \Phi) = \bigcap ((\lambda \varphi. \llbracket \varphi \rrbracket_{TESL}) ' \text{set } \Phi') \rangle$

by (*simp add: assms*)

ultimately show *?thesis* **using** *TESL-interp-remdups-absorb* **by** *auto*

qed

theorem *TESL-interp-decreases-setinc*:

assumes $\langle \text{set } \Phi \subseteq \text{set } \Phi' \rangle$

shows $\langle \llbracket \Phi \rrbracket_{TESL} \supseteq \llbracket \Phi' \rrbracket_{TESL} \rangle$

proof –

obtain Φ_r **where** *decompose*: $\langle \text{set } (\Phi @ \Phi_r) = \text{set } \Phi' \rangle$ **using** *assms* **by** *auto*

have $\langle \text{set } (\Phi @ \Phi_r) = \text{set } \Phi' \rangle$ **using** *assms decompose* **by** *blast*

moreover have $\langle (set \ \Phi) \cup (set \ \Phi_r) = set \ \Phi' \rangle$ **using** *assms decompose by auto*
 moreover have $\langle \llbracket \Phi' \rrbracket_{TESL} = \llbracket \Phi @ \Phi_r \rrbracket_{TESL} \rangle$ **using** *TESL-interp-set-lifting*
decompose by blast
 moreover have $\langle \llbracket \Phi @ \Phi_r \rrbracket_{TESL} = \llbracket \Phi \rrbracket_{TESL} \cap \llbracket \Phi_r \rrbracket_{TESL} \rangle$ **by** (*simp*
add: TESL-interp-homo-append)
 moreover have $\langle \llbracket \Phi \rrbracket_{TESL} \supseteq \llbracket \Phi \rrbracket_{TESL} \cap \llbracket \Phi_r \rrbracket_{TESL} \rangle$ **by** *simp*
 ultimately show *?thesis* **by** *simp*
qed

lemma *TESL-interp-decreases-add-head*:

assumes $\langle set \ \Phi \subseteq set \ \Phi' \rangle$
 shows $\langle \llbracket \varphi \# \Phi \rrbracket_{TESL} \supseteq \llbracket \varphi \# \Phi' \rrbracket_{TESL} \rangle$
using *assms TESL-interp-decreases-setinc by auto*

lemma *TESL-interp-decreases-add-tail*:

assumes $\langle set \ \Phi \subseteq set \ \Phi' \rangle$
 shows $\langle \llbracket \Phi @ [\varphi] \rrbracket_{TESL} \supseteq \llbracket \Phi' @ [\varphi] \rrbracket_{TESL} \rangle$
using *TESL-interp-decreases-setinc[OF assms]*
by (*simp add: TESL-interpretation-image dual-order.trans*)

lemma *TESL-interp-absorb1*:

assumes $\langle set \ \Phi_1 \subseteq set \ \Phi_2 \rangle$
 shows $\langle \llbracket \Phi_1 @ \Phi_2 \rrbracket_{TESL} = \llbracket \Phi_2 \rrbracket_{TESL} \rangle$
by (*simp add: Int-absorb1 TESL-interp-decreases-setinc TESL-interp-homo-append*
assms)

lemma *TESL-interp-absorb2*:

assumes $\langle set \ \Phi_2 \subseteq set \ \Phi_1 \rangle$
 shows $\langle \llbracket \Phi_1 @ \Phi_2 \rrbracket_{TESL} = \llbracket \Phi_1 \rrbracket_{TESL} \rangle$
using *TESL-interp-absorb1 TESL-interp-commute assms by blast*

3.5 Some special cases

lemma *NoSporadic-stable [simp]*:

$\langle \llbracket \Phi \rrbracket_{TESL} \subseteq \llbracket NoSporadic \ \Phi \rrbracket_{TESL} \rangle$
proof –
 from *filter-is-subset* **have** $\langle set \ (NoSporadic \ \Phi) \subseteq set \ \Phi \rangle$.
 from *TESL-interp-decreases-setinc[OF this]* **show** *?thesis* .
qed

lemma *NoSporadic-idem [simp]*:

$\langle \llbracket \Phi \rrbracket_{TESL} \cap \llbracket NoSporadic \ \Phi \rrbracket_{TESL} = \llbracket \Phi \rrbracket_{TESL} \rangle$
using *NoSporadic-stable by blast*

lemma *NoSporadic-setinc*:

$\langle set \ (NoSporadic \ \Phi) \subseteq set \ \Phi \rangle$
by (*rule filter-is-subset*)

end

```
theory SymbolicPrimitive
  imports Run
```

```
begin
datatype cnt-expr =
  TickCountLess ⟨clock⟩ ⟨instant-index⟩ (#<)
| TickCountLeq ⟨clock⟩ ⟨instant-index⟩ (#≤)
```

3.5.1 Symbolic Primitives for Runs

```
datatype tag-var =
  TSchematic ⟨clock * instant-index⟩ (τvar)

datatype 'τ constr =
  Timestamp      ⟨clock⟩ ⟨instant-index⟩ ⟨'τ tag-const⟩      (- ↓ - @ -)
| TimeDelay      ⟨clock⟩ ⟨instant-index⟩ ⟨'τ tag-const⟩ ⟨clock⟩ (- @ - ⊕ - ⇒ -)
| Ticks          ⟨clock⟩ ⟨instant-index⟩                      (- ↑ -)
| NotTicks       ⟨clock⟩ ⟨instant-index⟩                      (- ¬↑ -)
| NotTicksUntil  ⟨clock⟩ ⟨instant-index⟩                      (- ¬↑ < -)
| NotTicksFrom   ⟨clock⟩ ⟨instant-index⟩                      (- ¬↑ ≥ -)
| TagArith       ⟨tag-var⟩ ⟨tag-var⟩ ⟨('τ tag-const × 'τ tag-const) ⇒ bool⟩ ([-, -] ∈ -)
| TickCntArith   ⟨cnt-expr⟩ ⟨cnt-expr⟩ ⟨(nat × nat) ⇒ bool⟩ ([-, -] ∈ -)
| TickCntLeq     ⟨cnt-expr⟩ ⟨cnt-expr⟩                      (- ≤ -)
```

```
type-synonym 'τ system = ⟨'τ constr list⟩
```

— The abstract machine follows the intuition: past [$@term\Gamma$], current index [n], present [$@term\Psi$], future [$@term\Phi$] Beware: This type is slightly different from the one originally implemented in Heron

```
type-synonym 'τ config = ⟨'τ system * instant-index * 'τ TESL-formula * 'τ
  TESL-formula⟩
```

3.6 Semantics of Primitive Constraints

```
fun counter-expr-eval :: ⟨('τ::linordered-field) run ⇒ cnt-expr ⇒ nat⟩ ([ - ⊢ - ]cntexpr)
```

where

```
⟨[ ρ ⊢ #< clk indx ]cntexpr = run-tick-count-strictly ρ clk indx⟩
| ⟨[ ρ ⊢ #≤ clk indx ]cntexpr = run-tick-count ρ clk indx⟩
```

```
fun symbolic-run-interpretation-primitive
```

```
::⟨('τ::linordered-field) constr ⇒ 'τ run set⟩ ([ - ]prim)
```

where

```
⟨[ K ↑ n ]prim = { ρ. hamlet ((Rep-run ρ) n K) }⟩
| ⟨[ K @ n0 ⊕ δt ⇒ K' ]prim = { ρ. ∀ n ≥ n0. first-time ρ K n (time ((Rep-run
  ρ) n0 K) + δt)
  ⟶ hamlet ((Rep-run ρ) n K) }⟩
```


$$\begin{aligned}
| \langle \llbracket K \neg\uparrow n \rrbracket_{\text{prim}} &= \{ \varrho. \neg \text{hamlet} ((\text{Rep-run } \varrho) \ n \ K) \} \\
| \langle \llbracket K \neg\uparrow < n \rrbracket_{\text{prim}} &= \{ \varrho. \forall i < n. \neg \text{hamlet} ((\text{Rep-run } \varrho) \ i \ K) \} \\
| \langle \llbracket K \neg\uparrow \geq n \rrbracket_{\text{prim}} &= \{ \varrho. \forall i \geq n. \neg \text{hamlet} ((\text{Rep-run } \varrho) \ i \ K) \} \\
| \langle \llbracket K \Downarrow n @ \tau \rrbracket_{\text{prim}} &= \{ \varrho. \text{time} ((\text{Rep-run } \varrho) \ n \ K) = \tau \} \\
| \langle \llbracket [\tau_{\text{var}}(K_1, n_1), \tau_{\text{var}}(K_2, n_2)] \in R \rrbracket_{\text{prim}} &= \\
&\quad \{ \varrho. R (\text{time} ((\text{Rep-run } \varrho) \ n_1 \ K_1), \text{time} ((\text{Rep-run } \varrho) \ n_2 \ K_2)) \} \\
| \langle \llbracket [e_1, e_2] \in R \rrbracket_{\text{prim}} &= \{ \varrho. R (\llbracket \varrho \vdash e_1 \rrbracket_{\text{cntexpr}}, \llbracket \varrho \vdash e_2 \rrbracket_{\text{cntexpr}}) \} \\
| \langle \llbracket \text{cnt-}e_1 \preceq \text{cnt-}e_2 \rrbracket_{\text{prim}} &= \{ \varrho. \llbracket \varrho \vdash \text{cnt-}e_1 \rrbracket_{\text{cntexpr}} \leq \llbracket \varrho \vdash \text{cnt-}e_2 \rrbracket_{\text{cntexpr}} \}
\end{aligned}$$

fun *symbolic-run-interpretation*

$:: \langle (\tau::\text{linordered-field}) \text{ constr list} \Rightarrow (\tau::\text{linordered-field}) \text{ run set} \rangle (\llbracket - \rrbracket_{\text{prim}})$

where

$\langle \llbracket [] \rrbracket_{\text{prim}} = \{ -. \text{True} \} \rangle$

$| \langle \llbracket \llbracket \gamma \# \Gamma \rrbracket_{\text{prim}} = \llbracket \gamma \rrbracket_{\text{prim}} \cap \llbracket \Gamma \rrbracket_{\text{prim}} \rangle$

lemma *symbolic-run-interp-cons-morph*:

$\langle \llbracket \gamma \rrbracket_{\text{prim}} \cap \llbracket \Gamma \rrbracket_{\text{prim}} = \llbracket \llbracket \gamma \# \Gamma \rrbracket_{\text{prim}} \rangle$

by *auto*

definition *consistent-context* $:: \langle (\tau::\text{linordered-field}) \text{ constr list} \Rightarrow \text{bool} \rangle$

where

$\langle \text{consistent-context } \Gamma \equiv \exists \varrho. \varrho \in \llbracket \Gamma \rrbracket_{\text{prim}} \rangle$

3.6.1 Defining a method for witness construction

— Initial states

abbreviation *initial-run* $:: \langle (\tau::\text{linordered-field}) \text{ run} \rangle (\varrho_{\odot})$ **where**

$\langle \varrho_{\odot} \equiv \text{Abs-run } ((\lambda -. (\text{False}, \tau_{\text{cst}} \ 0)) :: \text{nat} \Rightarrow \text{clock} \Rightarrow (\text{bool} \times \tau \text{ tag-const})) \rangle$

— To ensure monotonicity, time tag is set at a specific instant and forever after (stuttering)

fun *time-update*

$:: \langle \text{nat} \Rightarrow \text{clock} \Rightarrow (\tau::\text{linordered-field}) \text{ tag-const} \Rightarrow (\text{nat} \Rightarrow \text{clock} \Rightarrow (\text{bool} \times \tau \text{ tag-const})) \rangle$

$\Rightarrow (\text{nat} \Rightarrow \text{clock} \Rightarrow (\text{bool} \times \tau \text{ tag-const})) \rangle$

where

$\langle \text{time-update } n \ K \ \tau \ \varrho = (\lambda n' \ K'. \text{if } K = K' \wedge n \leq n' \text{ then } (\text{hamlet } (\varrho \ n \ K), \tau) \text{ else } \varrho \ n' \ K') \rangle$

3.7 Rules and properties of consistence

lemma *context-consistency-preservationI*:

$\langle \text{consistent-context } ((\gamma :: (\tau::\text{linordered-field}) \text{ constr}) \# \Gamma) \Rightarrow \text{consistent-context } \Gamma \rangle$

unfolding *consistent-context-def*

by *auto*

— This is very restrictive

inductive *context-independency* $:: \langle (\tau::\text{linordered-field}) \text{ constr} \Rightarrow \tau \text{ constr list} \Rightarrow$

bool $\langle - \bowtie - \rangle$

where

NotTicks-independency:

$$\langle (K \uparrow n) \notin \text{set } \Gamma \implies (K \neg\uparrow n) \bowtie \Gamma \rangle$$

| *Ticks-independency:*

$$\langle (K \neg\uparrow n) \notin \text{set } \Gamma \implies (K \uparrow n) \bowtie \Gamma \rangle$$

| *Timestamp-independency:*

$$\langle \nexists \tau'. \tau' = \tau \wedge (K \Downarrow n @ \tau) \in \text{set } \Gamma \implies (K \Downarrow n @ \tau) \bowtie \Gamma \rangle$$

~~*lemma context-consistency-preservationE: assumes consist: consistent-context Γ and indepen: $\forall \gamma \in \Gamma$. shows consistent-context $(\# \Gamma)$ oops*~~

3.8 Major Theorems

3.8.1 Fixpoint lemma

theorem *symrun-interp-fixpoint:*

$$\langle \bigcap ((\lambda \gamma. \llbracket \gamma \rrbracket_{\text{prim}}) \text{ ` set } \Gamma) = \llbracket \Gamma \rrbracket_{\text{prim}} \rangle$$

proof (*induct* Γ)

case *Nil* thus ?case by *simp*

next

case *Cons* thus ?case by *auto*

qed

3.8.2 Expansion law

Similar to the expansion laws of lattices

theorem *symrun-interp-expansion:*

$$\langle \llbracket \Gamma_1 @ \Gamma_2 \rrbracket_{\text{prim}} = \llbracket \Gamma_1 \rrbracket_{\text{prim}} \cap \llbracket \Gamma_2 \rrbracket_{\text{prim}} \rangle$$

by (*induction* Γ_1 , *auto*)

3.9 Equational laws for TESL formulae denotationally interpreted

3.9.1 General laws

lemma *symrun-interp-assoc:*

$$\langle \llbracket (\Gamma_1 @ \Gamma_2) @ \Gamma_3 \rrbracket_{\text{prim}} = \llbracket \Gamma_1 @ (\Gamma_2 @ \Gamma_3) \rrbracket_{\text{prim}} \rangle$$

by *auto*

lemma *symrun-interp-commute:*

$$\langle \llbracket \Gamma_1 @ \Gamma_2 \rrbracket_{\text{prim}} = \llbracket \Gamma_2 @ \Gamma_1 \rrbracket_{\text{prim}} \rangle$$

by (*simp add: symrun-interp-expansion inf-sup-aci(1)*)

lemma *symrun-interp-left-commute:*

$$\langle \llbracket \Gamma_1 @ (\Gamma_2 @ \Gamma_3) \rrbracket_{\text{prim}} = \llbracket \Gamma_2 @ (\Gamma_1 @ \Gamma_3) \rrbracket_{\text{prim}} \rangle$$

unfolding *symrun-interp-expansion* **by** *auto*

lemma *symrun-interp-idem*:

$$\langle \llbracket \Gamma @ \Gamma \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}} \rangle$$

using *symrun-interp-expansion* **by** *auto*

lemma *symrun-interp-left-idem*:

$$\langle \llbracket \Gamma_1 @ (\Gamma_1 @ \Gamma_2) \rrbracket_{\text{prim}} = \llbracket \Gamma_1 @ \Gamma_2 \rrbracket_{\text{prim}} \rangle$$

using *symrun-interp-expansion* **by** *auto*

lemma *symrun-interp-right-idem*:

$$\langle \llbracket (\Gamma_1 @ \Gamma_2) @ \Gamma_2 \rrbracket_{\text{prim}} = \llbracket \Gamma_1 @ \Gamma_2 \rrbracket_{\text{prim}} \rangle$$

unfolding *symrun-interp-expansion* **by** *auto*

lemmas *symrun-interp-aci* = *symrun-interp-commute*

symrun-interp-assoc

symrun-interp-left-commute

symrun-interp-left-idem

— Identity element

lemma *symrun-interp-neutral1*:

$$\langle \llbracket [] @ \Gamma \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}} \rangle$$

by *simp*

lemma *symrun-interp-neutral2*:

$$\langle \llbracket \Gamma @ [] \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}} \rangle$$

by *simp*

3.9.2 Decreasing interpretation of TESL formulae

lemma *TESL-sem-decreases-head*:

$$\langle \llbracket \Gamma \rrbracket_{\text{prim}} \supseteq \llbracket \gamma \# \Gamma \rrbracket_{\text{prim}} \rangle$$

by *simp*

lemma *TESL-sem-decreases-tail*:

$$\langle \llbracket \Gamma \rrbracket_{\text{prim}} \supseteq \llbracket \Gamma @ [\gamma] \rrbracket_{\text{prim}} \rangle$$

by (*simp add: symrun-interp-expansion*)

lemma *symrun-interp-formula-stuttering*:

assumes $\langle \gamma \in \text{set } \Gamma \rangle$

shows $\langle \llbracket \gamma \# \Gamma \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}} \rangle$

proof —

have $\langle \gamma \# \Gamma = [\gamma] @ \Gamma \rangle$ **by** *simp*

hence $\langle \llbracket \gamma \# \Gamma \rrbracket_{\text{prim}} = \llbracket [\gamma] \rrbracket_{\text{prim}} \cap \llbracket \Gamma \rrbracket_{\text{prim}} \rangle$ **using** *symrun-interp-expansion*

by *simp*

thus *?thesis* **using** *assms symrun-interp-fixpoint* **by** *fastforce*

qed

lemma *symrun-interp-decreases*:

$$\langle \llbracket \Gamma \rrbracket_{\text{prim}} \supseteq \llbracket \gamma \# \Gamma \rrbracket_{\text{prim}} \rangle$$

by (*rule TESL-sem-decreases-head*)

lemma *symrun-interp-remdups-absorb*:

$\langle \llbracket \Gamma \rrbracket_{\text{prim}} = \llbracket \text{remdups } \Gamma \rrbracket_{\text{prim}} \rangle$

proof (*induct* Γ)

case *Nil* **thus** ?*case* **by** *simp*

next

case *Cons*

thus ?*case* **using** *symrun-interp-formula-stuttering* **by** *auto*

qed

lemma *symrun-interp-set-lifting*:

assumes $\langle \text{set } \Gamma = \text{set } \Gamma' \rangle$

shows $\langle \llbracket \Gamma \rrbracket_{\text{prim}} = \llbracket \Gamma' \rrbracket_{\text{prim}} \rangle$

proof –

have $\langle \text{set } (\text{remdups } \Gamma) = \text{set } (\text{remdups } \Gamma') \rangle$

by (*simp add: assms*)

moreover have $\text{fixpt}\Gamma: \langle \bigcap ((\lambda\gamma. \llbracket \gamma \rrbracket_{\text{prim}}) \text{ ‘ set } \Gamma) = \llbracket \Gamma \rrbracket_{\text{prim}} \rangle$

by (*simp add: symrun-interp-fixpoint*)

moreover have $\text{fixpt}\Gamma': \langle \bigcap ((\lambda\gamma. \llbracket \gamma \rrbracket_{\text{prim}}) \text{ ‘ set } \Gamma') = \llbracket \Gamma' \rrbracket_{\text{prim}} \rangle$

by (*simp add: symrun-interp-fixpoint*)

moreover have $\langle \bigcap ((\lambda\gamma. \llbracket \gamma \rrbracket_{\text{prim}}) \text{ ‘ set } \Gamma) = \bigcap ((\lambda\gamma. \llbracket \gamma \rrbracket_{\text{prim}}) \text{ ‘ set } \Gamma') \rangle$

by (*simp add: assms*)

ultimately show ?*thesis* **using** *symrun-interp-remdups-absorb* **by** *auto*

qed

theorem *symrun-interp-decreases-setinc*:

assumes $\langle \text{set } \Gamma \subseteq \text{set } \Gamma' \rangle$

shows $\langle \llbracket \Gamma \rrbracket_{\text{prim}} \supseteq \llbracket \Gamma' \rrbracket_{\text{prim}} \rangle$

proof –

obtain Γ_r **where** *decompose*: $\langle \text{set } (\Gamma @ \Gamma_r) = \text{set } \Gamma' \rangle$ **using** *assms* **by** *auto*

have $\langle \text{set } (\Gamma @ \Gamma_r) = \text{set } \Gamma' \rangle$ **using** *assms decompose* **by** *blast*

moreover have $\langle (\text{set } \Gamma) \cup (\text{set } \Gamma_r) = \text{set } \Gamma' \rangle$ **using** *assms decompose* **by** *auto*

moreover have $\langle \llbracket \Gamma' \rrbracket_{\text{prim}} = \llbracket \Gamma @ \Gamma_r \rrbracket_{\text{prim}} \rangle$ **using** *symrun-interp-set-lifting decompose* **by** *blast*

moreover have $\langle \llbracket \Gamma @ \Gamma_r \rrbracket_{\text{prim}} = \llbracket \Gamma \rrbracket_{\text{prim}} \cap \llbracket \Gamma_r \rrbracket_{\text{prim}} \rangle$ **by** (*simp add: symrun-interp-expansion*)

moreover have $\langle \llbracket \Gamma \rrbracket_{\text{prim}} \supseteq \llbracket \Gamma \rrbracket_{\text{prim}} \cap \llbracket \Gamma_r \rrbracket_{\text{prim}} \rangle$ **by** *simp*

ultimately show ?*thesis* **by** *simp*

qed

lemma *symrun-interp-decreases-add-head*:

assumes $\langle \text{set } \Gamma \subseteq \text{set } \Gamma' \rangle$

shows $\langle \llbracket \gamma \# \Gamma \rrbracket_{\text{prim}} \supseteq \llbracket \gamma \# \Gamma' \rrbracket_{\text{prim}} \rangle$

using *symrun-interp-decreases-setinc assms* **by** *auto*

lemma *symrun-interp-decreases-add-tail*:

assumes $\langle \text{set } \Gamma \subseteq \text{set } \Gamma' \rangle$

shows $\langle \llbracket \Gamma @ [\gamma] \rrbracket_{\text{prim}} \supseteq \llbracket \Gamma' @ [\gamma] \rrbracket_{\text{prim}} \rangle$

proof –

3.9. EQUATIONAL LAWS FOR TESL FORMULAE DENOTATIONALLY INTERPRETED29

from *symrun-interp-decreases-setinc*[*OF assms*] **have** $\langle \llbracket \Gamma' \rrbracket_{prim} \subseteq \llbracket \Gamma \rrbracket_{prim} \rangle$
 .
thus *?thesis* **by** (*simp add: symrun-interp-expansion dual-order.trans*)
qed

lemma *symrun-interp-absorb1*:
assumes $\langle set \ \Gamma_1 \subseteq set \ \Gamma_2 \rangle$
shows $\langle \llbracket \Gamma_1 @ \Gamma_2 \rrbracket_{prim} = \llbracket \Gamma_2 \rrbracket_{prim} \rangle$
by (*simp add: Int-absorb1 symrun-interp-decreases-setinc symrun-interp-expansion assms*)

lemma *symrun-interp-absorb2*:
assumes $\langle set \ \Gamma_2 \subseteq set \ \Gamma_1 \rangle$
shows $\langle \llbracket \Gamma_1 @ \Gamma_2 \rrbracket_{prim} = \llbracket \Gamma_1 \rrbracket_{prim} \rangle$
using *symrun-interp-absorb1 symrun-interp-commute assms* **by** *blast*
end

Chapter 4

Operational Semantics

```
theory Operational
imports
  SymbolicPrimitive
```

```
begin
```

4.1 Operational steps

abbreviation *uncurry-conf*

```
::('τ::linordered-field) system ⇒ instant-index ⇒ 'τ TESL-formula ⇒ 'τ TESL-formula
⇒ 'τ config⟩ (·, · ⊢ · ▷ · 80)
```

where

```
⟨Γ, n ⊢ Ψ ▷ Φ ≡ (Γ, n, Ψ, Φ)⟩
```

inductive *operational-semantics-intro*

```
::('τ::linordered-field) config ⇒ 'τ config ⇒ bool⟩ (· ⇨i · 70)
```

where

instant-i:

```
⟨Γ, n ⊢ [] ▷ Φ) ⇨i (Γ, Suc n ⊢ Φ ▷ [])⟩
```

inductive *operational-semantics-elim*

```
::('τ::linordered-field) config ⇒ 'τ config ⇒ bool⟩ (· ⇨e · 70)
```

where

sporadic-on-e1:

```
⟨Γ, n ⊢ ((K1 sporadic τ on K2) # Ψ) ▷ Φ)
⇨e (Γ, n ⊢ Ψ ▷ ((K1 sporadic τ on K2) # Φ))⟩
```

| *sporadic-on-e2:*

```
⟨Γ, n ⊢ ((K1 sporadic τ on K2) # Ψ) ▷ Φ)
⇨e (((K1 ↑ n) # (K2 ↓ n @ τ) # Γ), n ⊢ Ψ ▷ Φ)⟩
```

| *tagrel-e:*

```
⟨Γ, n ⊢ ((time-relation [K1, K2] ∈ R) # Ψ) ▷ Φ)
⇨e ((([τvar(K1, n), τvar(K2, n)] ∈ R) # Γ), n ⊢ Ψ ▷ ((time-relation [K1,
K2] ∈ R) # Φ))⟩
```

| *implies-e1:*

$\langle \Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow_e \langle ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \rangle$
| *implies-e2*:
 $\langle \Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow_e \langle ((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \rangle$
| *implies-not-e1*:
 $\langle \Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow_e \langle ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \rangle$
| *implies-not-e2*:
 $\langle \Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow_e \langle ((K_1 \uparrow n) \# (K_2 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \rangle$
| *timedelayed-e1*:
 $\langle \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow_e \langle ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rangle$
| *timedelayed-e2*:
 $\langle \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow_e \langle ((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rangle$
| *weakly-precedes-e*:
 $\langle \Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow_e \langle ((\lceil \# \leq K_2 n, \# \leq K_1 n \rceil \in (\lambda(x,y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \rangle$
| *strictly-precedes-e*:
 $\langle \Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow_e \langle ((\lceil \# \leq K_2 n, \# < K_1 n \rceil \in (\lambda(x,y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi) \rangle$
| *kills-e1*:
 $\langle \Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow_e \langle ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rangle$
| *kills-e2*:
 $\langle \Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow_e \langle ((K_1 \uparrow n) \# (K_2 \neg\uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rangle$

inductive operational-semantics-step

$:: \langle (' \tau :: \text{linordered-field}) \text{ config} \Rightarrow ' \tau \text{ config} \Rightarrow \text{bool} \rangle \quad (- \hookrightarrow - \text{ } 70)$

where

intro-part:

$\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle \hookrightarrow_i \langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle$
 $\implies \langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle \hookrightarrow \langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle$

elims-part:

$\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle \hookrightarrow_e \langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle$
 $\implies \langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle \hookrightarrow \langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle$

abbreviation operational-semantics-step-rtranclp

$:: \langle (' \tau :: \text{linordered-field}) \text{ config} \Rightarrow ' \tau \text{ config} \Rightarrow \text{bool} \rangle \quad (- \hookrightarrow^{**} - \text{ } 70)$

where

$\langle \mathcal{C}_1 \hookrightarrow^{**} \mathcal{C}_2 \equiv \text{operational-semantics-step}^{**} \mathcal{C}_1 \mathcal{C}_2 \rangle$

abbreviation *operational-semantic-step-transclp*
 $::(\tau::\text{linordered-field}) \text{ config} \Rightarrow \tau \text{ config} \Rightarrow \text{bool}$ ($- \hookrightarrow^{++}$ - 70)

where

$\langle \mathcal{C}_1 \hookrightarrow^{++} \mathcal{C}_2 \equiv \text{operational-semantic-step}^{++} \mathcal{C}_1 \mathcal{C}_2 \rangle$

abbreviation *operational-semantic-step-reflclp*
 $::(\tau::\text{linordered-field}) \text{ config} \Rightarrow \tau \text{ config} \Rightarrow \text{bool}$ ($- \hookrightarrow^{==}$ - 70)

where

$\langle \mathcal{C}_1 \hookrightarrow^{==} \mathcal{C}_2 \equiv \text{operational-semantic-step}^{==} \mathcal{C}_1 \mathcal{C}_2 \rangle$

abbreviation *operational-semantic-step-relpowp*
 $::(\tau::\text{linordered-field}) \text{ config} \Rightarrow \text{nat} \Rightarrow \tau \text{ config} \Rightarrow \text{bool}$ ($- \hookrightarrow^+$ - 70)

where

$\langle \mathcal{C}_1 \hookrightarrow^n \mathcal{C}_2 \equiv (\text{operational-semantic-step} \hat{\wedge} n) \mathcal{C}_1 \mathcal{C}_2 \rangle$

definition *operational-semantic-elim-inv*
 $::(\tau::\text{linordered-field}) \text{ config} \Rightarrow \tau \text{ config} \Rightarrow \text{bool}$ ($- \hookrightarrow_e^{\leftarrow}$ - 70)

where

$\langle \mathcal{C}_1 \hookrightarrow_e^{\leftarrow} \mathcal{C}_2 \equiv \mathcal{C}_2 \hookrightarrow_e \mathcal{C}_1 \rangle$

4.2 Basic Lemmas

lemma *operational-semantic-trans-generalized:*

assumes $\langle \mathcal{C}_1 \hookrightarrow^n \mathcal{C}_2 \rangle$

assumes $\langle \mathcal{C}_2 \hookrightarrow^m \mathcal{C}_3 \rangle$

shows $\langle \mathcal{C}_1 \hookrightarrow^{n+m} \mathcal{C}_3 \rangle$

using *relcompp.relcompI*[of $\langle \text{operational-semantic-step} \hat{\wedge} n \rangle$ - -
 $\langle \text{operational-semantic-step} \hat{\wedge} m \rangle$, OF *assms*]

by (*simp add: relpowp-add*)

abbreviation *Cnext-solve*

$::(\tau::\text{linordered-field}) \text{ config} \Rightarrow \tau \text{ config set} \Rightarrow (\mathcal{C}_{\text{next}} -)$

where

$\langle \mathcal{C}_{\text{next}} \mathcal{S} \equiv \{ \mathcal{S}'. \mathcal{S} \hookrightarrow \mathcal{S}' \} \rangle$

lemma *Cnext-solve-instant:*

$\langle (\mathcal{C}_{\text{next}} (\Gamma, n \vdash [] \triangleright \Phi)) \supseteq \{ \Gamma, \text{Suc } n \vdash \Phi \triangleright [] \} \rangle$

by (*simp add: operational-semantic-step.simps operational-semantic-intro.instant-i*)

lemma *Cnext-solve-sporadicon:*

$\langle (\mathcal{C}_{\text{next}} (\Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi))$

$\supseteq \{ \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi), ((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi \} \rangle$

by (*simp add: operational-semantic-step.simps operational-semantic-elim.sporadic-on-e1 operational-semantic-elim.sporadic-on-e2*)

lemma *Cnext-solve-tagrel:*

$\langle (\mathcal{C}_{\text{next}} (\Gamma, n \vdash ((\text{time-relation } [K_1, K_2] \in R) \# \Psi) \triangleright \Phi))$

$\supseteq \{ ([\tau_{\text{var}}(K_1, n), \tau_{\text{var}}(K_2, n)] \in R) \# \Gamma, n \vdash \Psi \triangleright ((\text{time-relation } [K_1,$

$K_2] \in R) \# \Phi) \rangle$
by (*simp add: operational-semantic-step.simps operational-semantic-elim.tagrel-e*)

lemma *Cnext-solve-implies:*

$\langle (C_{next} (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi))$
 $\supseteq \{ ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi),$
 $((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \}$

by (*simp add: operational-semantic-step.simps operational-semantic-elim.implies-e1*
operational-semantic-elim.implies-e2)

lemma *Cnext-solve-implies-not:*

$\langle (C_{next} (\Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi))$
 $\supseteq \{ ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi),$
 $((K_1 \uparrow n) \# (K_2 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \}$

by (*simp add: operational-semantic-step.simps operational-semantic-elim.implies-not-e1*
operational-semantic-elim.implies-not-e2)

lemma *Cnext-solve-timedelayed:*

$\langle (C_{next} (\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi))$
 $\supseteq \{ ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3)$
 $\# \Phi),$
 $((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \}$

by (*simp add: operational-semantic-step.simps operational-semantic-elim.timedelayed-e1*
operational-semantic-elim.timedelayed-e2)

lemma *Cnext-solve-weakly-precedes:*

$\langle (C_{next} (\Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi))$
 $\supseteq \{ ((\lceil \#^{\leq} K_2 n, \#^{\leq} K_1 n \rceil \in (\lambda(x,y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ weakly}$
precedes $K_2) \# \Phi) \}$

by (*simp add: operational-semantic-step.simps operational-semantic-elim.weakly-precedes-e*)

lemma *Cnext-solve-strictly-precedes:*

$\langle (C_{next} (\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi))$
 $\supseteq \{ ((\lceil \#^{\leq} K_2 n, \#^{<} K_1 n \rceil \in (\lambda(x,y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly}$
precedes $K_2) \# \Phi) \}$

by (*simp add: operational-semantic-step.simps operational-semantic-elim.strictly-precedes-e*)

lemma *Cnext-solve-kills:*

$\langle (C_{next} (\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi))$
 $\supseteq \{ ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi),$
 $((K_1 \uparrow n) \# (K_2 \neg\uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \}$

by (*simp add: operational-semantic-step.simps operational-semantic-elim.kills-e1*
operational-semantic-elim.kills-e2)

lemma *empty-spec-reductions:*

$\langle ([], 0 \vdash [] \triangleright []) \hookrightarrow^k ([], k \vdash [] \triangleright []) \rangle$

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 5

Equivalence of Operational and Denotational Semantics

```

theory Corecursive-Prop
imports
  SymbolicPrimitive
  Operational
  Denotational

```

```

begin

```

5.1 Stepwise denotational interpretation of TESL atoms

Denotational interpretation of TESL bounded by index

```

fun TESL-interpretation-atomic-stepwise
  ::  $\langle (\tau :: \text{linordered-field}) \text{ TESL-atomic} \Rightarrow \text{nat} \Rightarrow \text{'}\tau \text{ run set'} \rangle (\llbracket - \rrbracket_{\text{TESL}}^{\geq})$ 
where
   $\langle \llbracket K_1 \text{ sporadic } \tau \text{ on } K_2 \rrbracket_{\text{TESL}}^{\geq i} =$ 
     $\{ \varrho. \exists n \geq i. \text{hamlet } ((\text{Rep-run } \varrho) \text{ } n \text{ } K_1) = \text{True} \wedge \text{time } ((\text{Rep-run } \varrho) \text{ } n \text{ } K_2)$ 
 $= \tau \} \rangle$ 
  |  $\langle \llbracket \text{time-relation } [K_1, K_2] \in R \rrbracket_{\text{TESL}}^{\geq i} =$ 
     $\{ \varrho. \forall n \geq i. R (\text{time } ((\text{Rep-run } \varrho) \text{ } n \text{ } K_1), \text{time } ((\text{Rep-run } \varrho) \text{ } n \text{ } K_2)) \} \rangle$ 
  |  $\langle \llbracket \text{master implies slave} \rrbracket_{\text{TESL}}^{\geq i} =$ 
     $\{ \varrho. \forall n \geq i. \text{hamlet } ((\text{Rep-run } \varrho) \text{ } n \text{ } \text{master}) \longrightarrow \text{hamlet } ((\text{Rep-run } \varrho) \text{ } n \text{ } \text{slave})$ 
 $\} \rangle$ 
  |  $\langle \llbracket \text{master implies not slave} \rrbracket_{\text{TESL}}^{\geq i} =$ 
     $\{ \varrho. \forall n \geq i. \text{hamlet } ((\text{Rep-run } \varrho) \text{ } n \text{ } \text{master}) \longrightarrow \neg \text{hamlet } ((\text{Rep-run } \varrho) \text{ } n \text{ } \text{slave}) \} \rangle$ 
  |  $\langle \llbracket \text{master time-delayed by } \delta\tau \text{ on measuring implies slave} \rrbracket_{\text{TESL}}^{\geq i} =$ 
     $\{ \varrho. \forall n \geq i. \text{hamlet } ((\text{Rep-run } \varrho) \text{ } n \text{ } \text{master}) \longrightarrow$ 
 $(\text{let measured-time} = \text{time } ((\text{Rep-run } \varrho) \text{ } n \text{ } \text{measuring}) \text{ in}$ 
 $\forall m \geq n. \text{first-time } \varrho \text{ measuring } m (\text{measured-time} + \delta\tau)$ 

```

$$\begin{aligned}
& \longrightarrow \text{hamlet } ((\text{Rep-run } \varrho) \text{ } m \text{ slave}) \\
&) \\
& \rangle \\
& | \langle \llbracket K_1 \text{ weakly precedes } K_2 \rrbracket_{TESL}^{\geq i} = \\
& \quad \{ \varrho. \forall n \geq i. (\text{run-tick-count } \varrho \text{ } K_2 \text{ } n) \leq (\text{run-tick-count } \varrho \text{ } K_1 \text{ } n) \} \rangle \\
& | \langle \llbracket K_1 \text{ strictly precedes } K_2 \rrbracket_{TESL}^{\geq i} = \\
& \quad \{ \varrho. \forall n \geq i. (\text{run-tick-count } \varrho \text{ } K_2 \text{ } n) \leq (\text{run-tick-count-strictly } \varrho \text{ } K_1 \text{ } n) \} \rangle \\
& | \langle \llbracket K_1 \text{ kills } K_2 \rrbracket_{TESL}^{\geq i} = \\
& \quad \{ \varrho. \forall n \geq i. \text{hamlet } ((\text{Rep-run } \varrho) \text{ } n \text{ } K_1) \longrightarrow (\forall m \geq n. \neg \text{hamlet } ((\text{Rep-run } \varrho) \\
& m \text{ } K_2)) \} \rangle
\end{aligned}$$

theorem predicate-Inter-unfold:

$$\begin{aligned}
& \langle \{ \varrho. \forall n. P \varrho n \} = \bigcap \{ Y. \exists n. Y = \{ \varrho. P \varrho n \} \} \rangle \\
& \text{by (simp add: Collect-all-eq full-SetCompr-eq)}
\end{aligned}$$

theorem predicate-Union-unfold:

$$\begin{aligned}
& \langle \{ \varrho. \exists n. P \varrho n \} = \bigcup \{ Y. \exists n. Y = \{ \varrho. P \varrho n \} \} \rangle \\
& \text{by auto}
\end{aligned}$$

lemma TESL-interp-unfold-stepwise-sporadicon:

$$\begin{aligned}
& \text{shows } \langle \llbracket K_1 \text{ sporadic } \tau \text{ on } K_2 \rrbracket_{TESL} = \bigcup \{ Y. \exists n::\text{nat}. Y = \llbracket K_1 \text{ sporadic } \tau \\
& \text{on } K_2 \rrbracket_{TESL}^{\geq n} \} \rangle \\
& \text{by auto}
\end{aligned}$$

lemma TESL-interp-unfold-stepwise-tagrelgen:

$$\begin{aligned}
& \text{shows } \langle \llbracket \text{time-relation } [K_1, K_2] \in R \rrbracket_{TESL} = \bigcap \{ Y. \exists n::\text{nat}. Y = \llbracket \text{time-relation } [K_1, K_2] \in R \rrbracket_{TESL}^{\geq n} \} \rangle \\
& \text{by auto}
\end{aligned}$$

lemma TESL-interp-unfold-stepwise-implies:

$$\begin{aligned}
& \text{shows } \langle \llbracket \text{master implies slave} \rrbracket_{TESL} = \bigcap \{ Y. \exists n::\text{nat}. Y = \llbracket \text{master implies} \\
& \text{slave} \rrbracket_{TESL}^{\geq n} \} \rangle \\
& \text{by auto}
\end{aligned}$$

lemma TESL-interp-unfold-stepwise-implies-not:

$$\begin{aligned}
& \text{shows } \langle \llbracket \text{master implies not slave} \rrbracket_{TESL} = \bigcap \{ Y. \exists n::\text{nat}. Y = \llbracket \text{master} \\
& \text{implies not slave} \rrbracket_{TESL}^{\geq n} \} \rangle \\
& \text{by auto}
\end{aligned}$$

lemma TESL-interp-unfold-stepwise-timedelayed:

$$\begin{aligned}
& \text{shows } \langle \llbracket \text{master time-delayed by } \delta\tau \text{ on measuring implies slave} \rrbracket_{TESL} \\
& = \bigcap \{ Y. \exists n::\text{nat}. Y = \llbracket \text{master time-delayed by } \delta\tau \text{ on measuring implies} \\
& \text{slave} \rrbracket_{TESL}^{\geq n} \} \rangle \\
& \text{by auto}
\end{aligned}$$

lemma TESL-interp-unfold-stepwise-weakly-precedes:

$$\begin{aligned}
& \text{shows } \langle \llbracket K_1 \text{ weakly precedes } K_2 \rrbracket_{TESL} = \bigcap \{ Y. \exists n::\text{nat}. Y = \llbracket K_1 \text{ weakly} \\
& \text{precedes } K_2 \rrbracket_{TESL}^{\geq n} \} \rangle
\end{aligned}$$

5.1. STEPWISE DENOTATIONAL INTERPRETATION OF TESL ATOMS 39

by *auto*

lemma *TESL-interp-unfold-stepwise-strictly-precedes*:

shows $\langle \llbracket K_1 \text{ strictly precedes } K_2 \rrbracket_{TESL} = \bigcap \{Y. \exists n::nat. Y = \llbracket K_1 \text{ strictly precedes } K_2 \rrbracket_{TESL}^{\geq n}\} \rangle$

by *auto*

lemma *TESL-interp-unfold-stepwise-kills*:

shows $\langle \llbracket \text{master kills slave} \rrbracket_{TESL} = \bigcap \{Y. \exists n::nat. Y = \llbracket \text{master kills slave} \rrbracket_{TESL}^{\geq n}\} \rangle$

by *auto*

theorem *TESL-interp-unfold-stepwise-positive-atoms*:

assumes $\langle \text{positive-atom } \varphi \rangle$

shows $\langle \llbracket \varphi::\tau::\text{linordered-field TESL-atomic} \rrbracket_{TESL} = \bigcup \{Y. \exists n::nat. Y = \llbracket \varphi \rrbracket_{TESL}^{\geq n}\} \rangle$

proof –

from *positive-atom.elims(2)[OF assms]*

obtain $u\ v\ w$ **where** $\langle \varphi = (u \text{ sporadic } v \text{ on } w) \rangle$ **by** *blast*

with *TESL-interp-unfold-stepwise-sporadicon* **show** *?thesis* **by** *simp*
qed

theorem *TESL-interp-unfold-stepwise-negative-atoms*:

assumes $\langle \neg \text{positive-atom } \varphi \rangle$

shows $\langle \llbracket \varphi \rrbracket_{TESL} = \bigcap \{Y. \exists n::nat. Y = \llbracket \varphi \rrbracket_{TESL}^{\geq n}\} \rangle$

proof (*cases* φ)

case *SporadicOn* **thus** *?thesis* **using** *assms* **by** *simp*

next

case (*TagRelation* $x41\ x42\ x43$)

thus *?thesis* **using** *TESL-interp-unfold-stepwise-tagrelgen* **by** *simp*

next

case (*Implies* $x51\ x52$)

thus *?thesis* **using** *TESL-interp-unfold-stepwise-implies* **by** *simp*

next

case (*ImpliesNot* $x51\ x52$)

thus *?thesis* **using** *TESL-interp-unfold-stepwise-implies-not* **by** *simp*

next

case (*TimeDelayedBy* $x61\ x62\ x63\ x64$)

thus *?thesis* **using** *TESL-interp-unfold-stepwise-timedelayed* **by** *simp*

next

case (*WeaklyPrecedes* $x61\ x62$)

then show *?thesis*

using *TESL-interp-unfold-stepwise-weakly-precedes* **by** *simp*

next

case (*StrictlyPrecedes* $x61\ x62$)

then show *?thesis*

using *TESL-interp-unfold-stepwise-strictly-precedes* **by** *simp*

next

case (*Kills* $x63\ x64$)

then show *?thesis*
 using *TESL-interp-unfold-stepwise-kills* by *simp*
 qed

lemma *forall-nat-expansion*:

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

proof –

have $\langle (\forall n \geq (n_0 :: \text{nat}). P\ n) = (\forall n. (n = n_0 \vee n > n_0) \longrightarrow P\ n) \rangle$ using *le-less*
 by *blast*

also have $\langle \dots = (P\ n_0 \wedge (\forall n > n_0. P\ n)) \rangle$ by *blast*

finally show *?thesis* using *Suc-le-eq* by *simp*

qed

lemma *exists-nat-expansion*:

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

proof –

have $\langle (\exists n \geq (n_0 :: \text{nat}). P\ n) = (\exists n. (n = n_0 \vee n > n_0) \wedge P\ n) \rangle$ using *le-less*
 by *blast*

also have $\langle \dots = (\exists n. (P\ n_0) \vee (n > n_0 \wedge P\ n)) \rangle$ by *blast*

finally show *?thesis* using *Suc-le-eq* by *simp*

qed

5.2 Coinduction Unfolding Properties

lemma *TESL-interp-stepwise-sporadicon-cst-coind-unfold*:

shows $\langle \llbracket K_1 \text{ sporadic } \tau \text{ on } K_2 \rrbracket_{TESL}^{\geq n} =$

$\llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket K_2 \downarrow n @ \tau \rrbracket_{prim}$
 $\cup \llbracket K_1 \text{ sporadic } \tau \text{ on } K_2 \rrbracket_{TESL}^{\geq \text{Suc } n_1}$

proof –

have $\langle \{ \varrho. \exists m \geq n. \text{hamlet } ((\text{Rep-run } \varrho)\ m\ K_1) = \text{True} \wedge \text{time } ((\text{Rep-run } \varrho)\ m\ K_2) = \tau \} \rangle$

$= \langle \{ \varrho. \text{hamlet } ((\text{Rep-run } \varrho)\ n\ K_1) = \text{True} \wedge \text{time } ((\text{Rep-run } \varrho)\ n\ K_2) = \tau$
 $\vee (\exists m \geq \text{Suc } n. \text{hamlet } ((\text{Rep-run } \varrho)\ m\ K_1) = \text{True} \wedge \text{time } ((\text{Rep-run } \varrho)\ m\ K_2) = \tau) \} \rangle$

using *Suc-leD not-less-eq-eq* by *fastforce*

moreover have $\langle \{ \varrho. \text{hamlet } ((\text{Rep-run } \varrho)\ n\ K_1) = \text{True} \wedge \text{time } ((\text{Rep-run } \varrho)\ n\ K_2) = \tau$

$\vee (\exists m \geq \text{Suc } n. \text{hamlet } ((\text{Rep-run } \varrho)\ m\ K_1) = \text{True} \wedge \text{time } ((\text{Rep-run } \varrho)\ m\ K_2) = \tau) \} \rangle$

$= \llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket K_2 \downarrow n @ \tau \rrbracket_{prim} \cup \llbracket K_1 \text{ sporadic } \tau \text{ on } K_2 \rrbracket_{TESL}^{\geq \text{Suc } n_1}$

by (*simp add: Collect-conj-eq Collect-disj-eq*)

ultimately show *?thesis* by *auto*

qed

lemma *TESL-interp-stepwise-sporadicon-coind-unfold*:

shows $\langle \llbracket K_1 \text{ sporadic } \tau \text{ on } K_2 \rrbracket_{TESL}^{\geq n} =$

$\llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket K_2 \downarrow n @ \tau \rrbracket_{prim}$

$\cup \llbracket K_1 \text{ sporadic } \tau \text{ on } K_2 \rrbracket_{TESL} \geq \text{Suc } n$
using *TESL-interp-stepwise-sporadicon-cst-coind-unfold* **by** *blast*

lemma *nat-set-suc*: $\langle \{x. \forall m \geq n. P \ x \ m\} = \{x. P \ x \ n\} \cap \{x. \forall m \geq \text{Suc } n. P \ x \ m\} \rangle$

proof

{ **fix** x
assume $h: \langle x \in \{x. \forall m \geq n. P \ x \ m\} \rangle$
hence $\langle P \ x \ n \rangle$ **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** $\langle \{x. \forall m \geq n. P \ x \ m\} \subseteq \{x. P \ x \ n\} \cap \{x. \forall m \geq \text{Suc } n. P \ x \ m\} \rangle$..

next

{ **fix** x
assume $h: \langle x \in \{x. P \ x \ n\} \cap \{x. \forall m \geq \text{Suc } n. P \ x \ m\} \rangle$
hence $\langle P \ x \ n \rangle$ **by** *simp*
moreover from h **have** $\langle \forall m \geq \text{Suc } n. P \ x \ m \rangle$ **by** *simp*
ultimately have $\langle \forall m \geq n. P \ x \ m \rangle$ **using** *forall-nat-expansion* **by** *blast*
hence $\langle x \in \{x. \forall m \geq n. P \ x \ m\} \rangle$ **by** *simp*
} **thus** $\langle \{x. P \ x \ n\} \cap \{x. \forall m \geq \text{Suc } n. P \ x \ m\} \subseteq \{x. \forall m \geq n. P \ x \ m\} \rangle$..

qed

lemma *TESL-interp-stepwise-tagrel-coind-unfold*:

shows $\llbracket \text{time-relation } [K_1, K_2] \in R \rrbracket_{TESL} \geq n =$
 $\llbracket [\tau_{var}(K_1, n), \tau_{var}(K_2, n)] \in R \rrbracket_{prim}$
 $\cap \llbracket \text{time-relation } [K_1, K_2] \in R \rrbracket_{TESL} \geq \text{Suc } n$

proof –

have $\langle \{ \varrho. \forall m \geq n. R \ (\text{time } ((\text{Rep-run } \varrho) \ m \ K_1), \text{time } ((\text{Rep-run } \varrho) \ m \ K_2)) \}$
 $= \{ \varrho. R \ (\text{time } ((\text{Rep-run } \varrho) \ n \ K_1), \text{time } ((\text{Rep-run } \varrho) \ n \ K_2)) \}$
 $\cap \{ \varrho. \forall m \geq \text{Suc } n. R \ (\text{time } ((\text{Rep-run } \varrho) \ m \ K_1), \text{time } ((\text{Rep-run } \varrho) \ m \ K_2)) \}$
 \rangle

using *nat-set-suc*[*of* $\langle n \rangle \langle \lambda x y. R \ (\text{time } ((\text{Rep-run } x) \ y \ K_1), \text{time } ((\text{Rep-run } x) \ y \ K_2)) \rangle$] **by** *simp*

then show *?thesis* **by** *auto*

qed

lemma *TESL-interp-stepwise-implies-coind-unfold*:

shows $\llbracket \text{master implies slave} \rrbracket_{TESL} \geq n =$
 $(\llbracket \text{master } \neg \uparrow n \rrbracket_{prim} \cup \llbracket \text{master } \uparrow n \rrbracket_{prim} \cap \llbracket \text{slave } \uparrow n \rrbracket_{prim})$
 $\cap \llbracket \text{master implies slave} \rrbracket_{TESL} \geq \text{Suc } n$

proof –

have $\langle \{ \varrho. \forall m \geq n. \text{hamlet } ((\text{Rep-run } \varrho) \ m \ \text{master}) \longrightarrow \text{hamlet } ((\text{Rep-run } \varrho) \ m \ \text{slave}) \}$

$= \{ \varrho. \text{hamlet } ((\text{Rep-run } \varrho) \ n \ \text{master}) \longrightarrow \text{hamlet } ((\text{Rep-run } \varrho) \ n \ \text{slave}) \}$
 $\cap \{ \varrho. \forall m \geq \text{Suc } n. \text{hamlet } ((\text{Rep-run } \varrho) \ m \ \text{master}) \longrightarrow \text{hamlet } ((\text{Rep-run } \varrho) \ m \ \text{slave}) \}$
 \rangle

using *nat-set-suc*[*of* $\langle n \rangle \langle \lambda x y. \text{hamlet } ((\text{Rep-run } x) \ y \ \text{master}) \longrightarrow \text{hamlet } ((\text{Rep-run } x) \ y \ \text{slave}) \rangle$] **by** *simp*

then show *?thesis* **by** *auto*

qed

lemma *TESL-interp-stepwise-implies-not-coind-unfold:*

shows $\llbracket \text{master implies not slave} \rrbracket_{TESL}^{\geq n} =$
 $(\llbracket \text{master} \neg \uparrow n \rrbracket_{prim} \cup \llbracket \text{master} \uparrow n \rrbracket_{prim} \cap \llbracket \text{slave} \neg \uparrow n \rrbracket_{prim})$
 $\cap \llbracket \text{master implies not slave} \rrbracket_{TESL}^{\geq \text{Suc } n}$

proof –

have $\langle \{ \varrho. \forall m \geq n. \text{hamlet } ((\text{Rep-run } \varrho) \ m \ \text{master}) \longrightarrow \neg \text{hamlet } ((\text{Rep-run } \varrho) \ m \ \text{slave}) \} \rangle$
 $= \{ \varrho. \text{hamlet } ((\text{Rep-run } \varrho) \ n \ \text{master}) \longrightarrow \neg \text{hamlet } ((\text{Rep-run } \varrho) \ n \ \text{slave}) \}$
 $\cap \{ \varrho. \forall m \geq \text{Suc } n. \text{hamlet } ((\text{Rep-run } \varrho) \ m \ \text{master}) \longrightarrow \neg \text{hamlet } ((\text{Rep-run } \varrho) \ m \ \text{slave}) \}$

using *nat-set-suc*[of $\langle n \rangle$ $\langle \lambda x y. \text{hamlet } ((\text{Rep-run } x) \ y \ \text{master}) \longrightarrow \neg \text{hamlet } ((\text{Rep-run } x) \ y \ \text{slave}) \rangle$] **by** *simp*

then show *?thesis* **by** *auto*

qed

lemma *TESL-interp-stepwise-timedelayed-coind-unfold:*

shows $\llbracket \text{master time-delayed by } \delta\tau \text{ on measuring implies slave} \rrbracket_{TESL}^{\geq n} =$
 $(\llbracket \text{master} \neg \uparrow n \rrbracket_{prim} \cup (\llbracket \text{master} \uparrow n \rrbracket_{prim} \cap \llbracket \text{measuring @ } n \oplus \delta\tau \Rightarrow \text{slave} \rrbracket_{prim}))$
 $\cap \llbracket \text{master time-delayed by } \delta\tau \text{ on measuring implies slave} \rrbracket_{TESL}^{\geq \text{Suc } n}$

proof –

let $?prop = \langle \lambda \varrho \ m. \text{hamlet } ((\text{Rep-run } \varrho) \ m \ \text{master}) \longrightarrow$
 $(\text{let measured-time} = \text{time } ((\text{Rep-run } \varrho) \ m \ \text{measuring}) \text{ in}$
 $\forall p \geq m. \text{first-time } \varrho \ \text{measuring } p \ (\text{measured-time} + \delta\tau)$
 $\longrightarrow \text{hamlet } ((\text{Rep-run } \varrho) \ p \ \text{slave})) \rangle$

have $\langle \{ \varrho. \forall m \geq n. ?prop \ \varrho \ m \} = \{ \varrho. ?prop \ \varrho \ n \} \cap \{ \varrho. \forall m \geq \text{Suc } n. ?prop \ \varrho \ m \} \rangle$

using *nat-set-suc*[of $\langle n \rangle$ $?prop$] **by** *blast*

also have $\langle \dots = \{ \varrho. ?prop \ \varrho \ n \} \cap \llbracket \text{master time-delayed by } \delta\tau \text{ on measuring implies slave} \rrbracket_{TESL}^{\geq \text{Suc } n} \rangle$ **by** *simp*

finally show *?thesis* **by** *auto*

qed

lemma *TESL-interp-stepwise-weakly-precedes-coind-unfold:*

shows $\llbracket K_1 \text{ weakly precedes } K_2 \rrbracket_{TESL}^{\geq n} =$
 $\llbracket ([\#^{\leq} K_2 \ n, \#^{\leq} K_1 \ n] \in (\lambda(x,y). x \leq y)) \rrbracket_{prim}$
 $\cap \llbracket K_1 \text{ weakly precedes } K_2 \rrbracket_{TESL}^{\geq \text{Suc } n}$

proof –

have $\langle \{ \varrho. \forall p \geq n. (\text{run-tick-count } \varrho \ K_2 \ p) \leq (\text{run-tick-count } \varrho \ K_1 \ p) \}$
 $= \{ \varrho. (\text{run-tick-count } \varrho \ K_2 \ n) \leq (\text{run-tick-count } \varrho \ K_1 \ n) \}$
 $\cap \{ \varrho. \forall p \geq \text{Suc } n. (\text{run-tick-count } \varrho \ K_2 \ p) \leq (\text{run-tick-count } \varrho \ K_1 \ p) \} \rangle$

using *nat-set-suc*[of $\langle n \rangle$ $\langle \lambda \varrho \ n. (\text{run-tick-count } \varrho \ K_2 \ n) \leq (\text{run-tick-count } \varrho \ K_1 \ n) \rangle$]

by *simp*

then show *?thesis* **by** *auto*

qed

lemma *TESL-interp-stepwise-strictly-precedes-coind-unfold:*

shows $\langle \llbracket K_1 \text{ strictly precedes } K_2 \rrbracket_{TESL}^{\geq n} =$
 $\llbracket (\#^{\leq} K_2 n, \#^{<} K_1 n) \in (\lambda(x,y). x \leq y) \rrbracket_{prim}$
 $\cap \llbracket K_1 \text{ strictly precedes } K_2 \rrbracket_{TESL}^{\geq \text{Suc } n} \rangle$

proof –

have $\langle \{ \varrho. \forall p \geq n. (\text{run-tick-count } \varrho K_2 p) \leq (\text{run-tick-count-strictly } \varrho K_1 p) \}$
 $= \{ \varrho. (\text{run-tick-count } \varrho K_2 n) \leq (\text{run-tick-count-strictly } \varrho K_1 n) \}$
 $\cap \{ \varrho. \forall p \geq \text{Suc } n. (\text{run-tick-count } \varrho K_2 p) \leq (\text{run-tick-count-strictly } \varrho K_1$
 $p) \} \rangle$

using *nat-set-suc*[*of* $\langle n \rangle \langle \lambda \varrho n. (\text{run-tick-count } \varrho K_2 n) \leq (\text{run-tick-count-strictly } \varrho K_1 n) \rangle$]

by *simp*

then show *?thesis* **by** *auto*

qed

lemma *TESL-interp-stepwise-kills-coind-unfold:*

shows $\langle \llbracket K_1 \text{ kills } K_2 \rrbracket_{TESL}^{\geq n} =$
 $(\llbracket K_1 \neg \uparrow n \rrbracket_{prim} \cup \llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket K_2 \neg \uparrow \geq n \rrbracket_{prim})$
 $\cap \llbracket K_1 \text{ kills } K_2 \rrbracket_{TESL}^{\geq \text{Suc } n} \rangle$

proof –

let *?kills* = $\langle \lambda n \varrho. \forall p \geq n. \text{hamlet } ((\text{Rep-run } \varrho) p K_1) \longrightarrow (\forall m \geq p. \neg \text{hamlet } ((\text{Rep-run } \varrho) m K_2)) \rangle$

let *?ticks* = $\langle \lambda n \varrho c. \text{hamlet } ((\text{Rep-run } \varrho) n c) \rangle$

let *?dead* = $\langle \lambda n \varrho c. \forall m \geq n. \neg \text{hamlet } ((\text{Rep-run } \varrho) m c) \rangle$

have $\langle \llbracket K_1 \text{ kills } K_2 \rrbracket_{TESL}^{\geq n} = \{ \varrho. ?kills n \varrho \} \rangle$ **by** *simp*

also have $\langle \dots = (\{ \varrho. \neg ?ticks n \varrho K_1 \} \cap \{ \varrho. ?kills (\text{Suc } n) \varrho \})$
 $\cup \{ \varrho. ?ticks n \varrho K_1 \} \cap \{ \varrho. ?dead n \varrho K_2 \} \rangle$

proof

{ fix $\varrho :: \langle \tau :: \text{linordered-field run} \rangle$

assume $\langle \varrho \in \{ \varrho. ?kills n \varrho \} \rangle$

hence $\langle ?kills n \varrho \rangle$ **by** *simp*

hence $\langle (?ticks n \varrho K_1 \wedge ?dead n \varrho K_2) \vee (\neg ?ticks n \varrho K_1 \wedge ?kills (\text{Suc } n)$

$\varrho) \rangle$

using *Suc-leD* **by** *blast*

hence $\langle \varrho \in (\{ \varrho. ?ticks n \varrho K_1 \} \cap \{ \varrho. ?dead n \varrho K_2 \})$

$\cup (\{ \varrho. \neg ?ticks n \varrho K_1 \} \cap \{ \varrho. ?kills (\text{Suc } n) \varrho \}) \rangle$

by *blast*

} thus $\langle \{ \varrho. ?kills n \varrho \}$

$\subseteq \{ \varrho. \neg ?ticks n \varrho K_1 \} \cap \{ \varrho. ?kills (\text{Suc } n) \varrho \}$

$\cup \{ \varrho. ?ticks n \varrho K_1 \} \cap \{ \varrho. ?dead n \varrho K_2 \} \rangle$ **by** *blast*

next

{ fix $\varrho :: \langle \tau :: \text{linordered-field run} \rangle$

assume $\langle \varrho \in (\{ \varrho. \neg ?ticks n \varrho K_1 \} \cap \{ \varrho. ?kills (\text{Suc } n) \varrho \})$

$\cup \{ \varrho. ?ticks n \varrho K_1 \} \cap \{ \varrho. ?dead n \varrho K_2 \} \rangle$

hence $\langle \neg ?ticks n \varrho K_1 \wedge ?kills (\text{Suc } n) \varrho$

$\vee ?ticks n \varrho K_1 \wedge ?dead n \varrho K_2 \rangle$ **by** *blast*

moreover have $\langle (\neg ?ticks n \varrho K_1 \wedge (?kills (\text{Suc } n) \varrho)) \longrightarrow ?kills n \varrho \rangle$

using *dual-order.antisym not-less-eq-eq* **by** *blast*

ultimately have $\langle ?kills n \varrho \vee ?ticks n \varrho K_1 \wedge ?dead n \varrho K_2 \rangle$ **by** *blast*

hence $\langle ?kills\ n\ \varrho \rangle$ using *le-trans* by *blast*
 } thus $\langle \{ \varrho. \neg ?ticks\ n\ \varrho\ K_1 \} \cap \{ \varrho. ?kills\ (Suc\ n)\ \varrho \}$
 $\cup \{ \varrho. ?ticks\ n\ \varrho\ K_1 \} \cap \{ \varrho. ?dead\ n\ \varrho\ K_2 \} \rangle$
 $\subseteq \{ \varrho. ?kills\ n\ \varrho \}$ by *blast*
 qed
 also have $\langle \dots = \{ \varrho. \neg ?ticks\ n\ \varrho\ K_1 \} \cap \{ \varrho. ?kills\ (Suc\ n)\ \varrho \}$
 $\cup \{ \varrho. ?ticks\ n\ \varrho\ K_1 \} \cap \{ \varrho. ?dead\ n\ \varrho\ K_2 \} \cap \{ \varrho. ?kills\ (Suc\ n)\ \varrho \} \rangle$
 using *Collect-cong* *Collect-disj-eq* by *auto*
 also have $\langle \dots = \llbracket K_1 \neg\uparrow n \rrbracket_{prim} \cap \llbracket K_1\ kills\ K_2 \rrbracket_{TESL}^{\geq\ Suc\ n}$
 $\cup \llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket K_2 \neg\uparrow \geq n \rrbracket_{prim} \cap \llbracket K_1\ kills\ K_2 \rrbracket_{TESL}^{\geq\ Suc\ n} \rangle$
 by *simp*
 finally show *?thesis* by *blast*
 qed

fun *TESL-interpretation-stepwise* :: $\langle ' \tau :: \text{linordered-field } \text{TESL-formula} \Rightarrow \text{nat} \Rightarrow$
 $' \tau \text{ run set} \rangle (\llbracket - \rrbracket_{TESL}^{\geq} \cdot) \text{ where}$
 $\langle \llbracket [] \rrbracket_{TESL}^{\geq n} = \{ \cdot. \text{True} \} \rangle$
 $| \langle \llbracket \varphi \# \Phi \rrbracket_{TESL}^{\geq n} = \llbracket \varphi \rrbracket_{TESL}^{\geq n} \cap \llbracket \Phi \rrbracket_{TESL}^{\geq n} \rangle$

lemma *TESL-interpretation-stepwise-fixpoint*:
 $\langle \llbracket \Phi \rrbracket_{TESL}^{\geq n} = \bigcap ((\lambda \varphi. \llbracket \varphi \rrbracket_{TESL}^{\geq n}) \cdot \text{set } \Phi) \rangle$
 by (*induction* Φ , *simp*, *auto*)

lemma *TESL-interpretation-stepwise-zero*:
 $\langle \llbracket \varphi \rrbracket_{TESL} = \llbracket \varphi \rrbracket_{TESL}^{\geq 0} \rangle$
 by (*induction* φ , *simp*+)

lemma *TESL-interpretation-stepwise-zero'*:
 $\langle \llbracket \Phi \rrbracket_{TESL} = \llbracket \Phi \rrbracket_{TESL}^{\geq 0} \rangle$
 by (*induction* Φ , *simp*, *simp add: TESL-interpretation-stepwise-zero*)

lemma *TESL-interpretation-stepwise-cons-morph*:
 $\langle \llbracket \varphi \rrbracket_{TESL}^{\geq n} \cap \llbracket \Phi \rrbracket_{TESL}^{\geq n} = \llbracket \varphi \# \Phi \rrbracket_{TESL}^{\geq n} \rangle$
 by *auto*

theorem *TESL-interp-stepwise-composition*:
 shows $\langle \llbracket \Phi_1 @ \Phi_2 \rrbracket_{TESL}^{\geq n} = \llbracket \Phi_1 \rrbracket_{TESL}^{\geq n} \cap \llbracket \Phi_2 \rrbracket_{TESL}^{\geq n} \rangle$
 by (*induction* Φ_1 , *simp*, *auto*)

5.3 Interpretation of configurations

fun *HeronConf-interpretation* :: $\langle ' \tau :: \text{linordered-field } \text{config} \Rightarrow ' \tau \text{ run set} \rangle (\llbracket - \rrbracket_{config}$
 $71) \text{ where}$
 $\langle \llbracket \Gamma, n \vdash \Psi \triangleright \Phi \rrbracket_{config} = \llbracket \Gamma \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL}^{\geq n} \cap \llbracket \Phi \rrbracket_{TESL}^{\geq Suc\ n} \rangle$

lemma *HeronConf-interp-composition*:
 shows $\langle \llbracket \Gamma_1, n \vdash \Psi_1 \triangleright \Phi_1 \rrbracket_{config} \cap \llbracket \Gamma_2, n \vdash \Psi_2 \triangleright \Phi_2 \rrbracket_{config}$
 $= \llbracket (\Gamma_1 @ \Gamma_2), n \vdash (\Psi_1 @ \Psi_2) \triangleright (\Phi_1 @ \Phi_2) \rrbracket_{config} \rangle$

using *TESL-interp-stepwise-composition symrun-interp-expansion*
 by (*simp add: TESL-interp-stepwise-composition symrun-interp-expansion inf-assoc*
inf-left-commute)

lemma *HeronConf-interp-stepwise-instant-cases:*

shows $\langle \llbracket \Gamma, n \vdash \Box \triangleright \Phi \rrbracket_{config} = \llbracket \Gamma, Suc\ n \vdash \Phi \triangleright \Box \rrbracket_{config} \rangle$
proof –
have $\langle \llbracket \Gamma, n \vdash \Box \triangleright \Phi \rrbracket_{config} = \llbracket \llbracket \Gamma \rrbracket_{prim} \cap \llbracket \Box \rrbracket_{TESL} \geq n \cap \llbracket \Phi \rrbracket_{TESL} \geq Suc\ n, \rangle$
by *simp*
moreover have $\langle \llbracket \Gamma, Suc\ n \vdash \Phi \triangleright \Box \rrbracket_{config} = \llbracket \llbracket \Gamma \rrbracket_{prim} \cap \llbracket \Phi \rrbracket_{TESL} \geq Suc\ n \cap \llbracket \Box \rrbracket_{TESL} \geq Suc\ n, \rangle$
by *simp*
moreover have $\langle \llbracket \llbracket \Gamma \rrbracket_{prim} \cap \llbracket \Box \rrbracket_{TESL} \geq n \cap \llbracket \Phi \rrbracket_{TESL} \geq Suc\ n = \llbracket \llbracket \Gamma \rrbracket_{prim} \cap \llbracket \Phi \rrbracket_{TESL} \geq Suc\ n \cap \llbracket \Box \rrbracket_{TESL} \geq Suc\ n, \rangle$
by *simp*
ultimately show *?thesis* **by** *blast*
qed

lemma *HeronConf-interp-stepwise-sporadicon-cases:*

shows $\langle \llbracket \Gamma, n \vdash ((K_1\ sporadic\ \tau\ on\ K_2) \# \Psi) \triangleright \Phi \rrbracket_{config} = \llbracket \Gamma, n \vdash \Psi \triangleright ((K_1\ sporadic\ \tau\ on\ K_2) \# \Phi) \rrbracket_{config} \cup \llbracket ((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi \rrbracket_{config} \rangle$
proof –
have $\langle \llbracket \Gamma, n \vdash (K_1\ sporadic\ \tau\ on\ K_2) \# \Psi \triangleright \Phi \rrbracket_{config} = \llbracket \llbracket \Gamma \rrbracket_{prim} \cap \llbracket (K_1\ sporadic\ \tau\ on\ K_2) \# \Psi \rrbracket_{TESL} \geq n \cap \llbracket \Phi \rrbracket_{TESL} \geq Suc\ n, \rangle$
by *simp*
moreover have $\langle \llbracket \Gamma, n \vdash \Psi \triangleright ((K_1\ sporadic\ \tau\ on\ K_2) \# \Phi) \rrbracket_{config} = \llbracket \llbracket \Gamma \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL} \geq n \cap \llbracket (K_1\ sporadic\ \tau\ on\ K_2) \# \Phi \rrbracket_{TESL} \geq Suc\ n, \rangle$
by *simp*
moreover have $\langle \llbracket ((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi \rrbracket_{config} = \llbracket ((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma) \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL} \geq n \cap \llbracket \Phi \rrbracket_{TESL} \geq Suc\ n, \rangle$
by *simp*
ultimately show *?thesis*
proof –
have $\langle (\llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket K_2 \downarrow n @ \tau \rrbracket_{prim} \cup \llbracket K_1\ sporadic\ \tau\ on\ K_2 \rrbracket_{TESL} \geq Suc\ n) \cap (\llbracket \llbracket \Gamma \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL} \geq n) = \llbracket K_1\ sporadic\ \tau\ on\ K_2 \rrbracket_{TESL} \geq n \cap (\llbracket \Psi \rrbracket_{TESL} \geq n \cap \llbracket \Gamma \rrbracket_{prim}) \rangle$
using *TESL-interp-stepwise-sporadicon-coind-unfold* **by** *blast*
then have $\langle \llbracket ((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma) \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL} \geq n \cup \llbracket \llbracket \Gamma \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL} \geq n \cap \llbracket K_1\ sporadic\ \tau\ on\ K_2 \rrbracket_{TESL} \geq Suc\ n = \llbracket (K_1\ sporadic\ \tau\ on\ K_2) \# \Psi \rrbracket_{TESL} \geq n \cap \llbracket \Gamma \rrbracket_{prim} \rangle$
by *auto*
then show *?thesis*
by *auto*
qed
qed

lemma *HeronConf-interp-stepwise-tagrel-cases:*

shows $\langle \llbracket \Gamma, n \vdash ((\text{time-relation } \lfloor K_1, K_2 \rfloor \in R) \# \Psi) \triangleright \Phi \rrbracket_{\text{config}}$
 $= \llbracket ((\lfloor \tau_{\text{var}}(K_1, n), \tau_{\text{var}}(K_2, n) \rfloor \in R) \# \Gamma), n \vdash \Psi \triangleright ((\text{time-relation } \lfloor K_1, K_2 \rfloor \in R) \# \Phi) \rrbracket_{\text{config}}$
proof –
have $\langle \llbracket \Gamma, n \vdash (\text{time-relation } \lfloor K_1, K_2 \rfloor \in R) \# \Psi \triangleright \Phi \rrbracket_{\text{config}} = \llbracket \llbracket \Gamma \rrbracket_{\text{prim}} \cap$
 $\llbracket (\text{time-relation } \lfloor K_1, K_2 \rfloor \in R) \# \Psi \rrbracket_{\text{TESL}} \geq n \cap \llbracket \llbracket \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n \rrbracket$
by simp
moreover have $\langle \llbracket ((\lfloor \tau_{\text{var}}(K_1, n), \tau_{\text{var}}(K_2, n) \rfloor \in R) \# \Gamma), n \vdash \Psi \triangleright ((\text{time-relation } \lfloor K_1, K_2 \rfloor \in R) \# \Phi) \rrbracket_{\text{config}}$
 $= \llbracket \llbracket (\lfloor \tau_{\text{var}}(K_1, n), \tau_{\text{var}}(K_2, n) \rfloor \in R) \# \Gamma \rrbracket_{\text{prim}} \cap \llbracket \llbracket \Psi \rrbracket_{\text{TESL}} \geq n \cap \llbracket (\text{time-relation } \lfloor K_1, K_2 \rfloor \in R) \# \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n \rrbracket$
by simp
ultimately show ?thesis
proof –
have $\langle \llbracket \lfloor \tau_{\text{var}}(K_1, n), \tau_{\text{var}}(K_2, n) \rfloor \in R \rrbracket_{\text{prim}} \cap \llbracket \text{time-relation } \lfloor K_1, K_2 \rfloor \in R \rrbracket_{\text{TESL}} \geq \text{Suc } n \cap \llbracket \llbracket \Psi \rrbracket_{\text{TESL}} \geq n = \llbracket (\text{time-relation } \lfloor K_1, K_2 \rfloor \in R) \# \Psi \rrbracket_{\text{TESL}} \geq n \rrbracket$
using TESL-interp-stepwise-tagrel-coind-unfold TESL-interpretation-stepwise-cons-morph
by blast
then show ?thesis
by auto
qed
qed

lemma *HeronConf-interp-stepwise-implies-cases:*

shows $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi \rrbracket_{\text{config}}$
 $= \llbracket ((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \rrbracket_{\text{config}}$
 $\cup \llbracket ((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \rrbracket_{\text{config}}$
proof –
have $\langle \llbracket \Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \triangleright \Phi \rrbracket_{\text{config}} = \llbracket \llbracket \Gamma \rrbracket_{\text{prim}} \cap \llbracket \llbracket (K_1 \text{ implies } K_2) \# \Psi \rrbracket_{\text{TESL}} \geq n \cap \llbracket \llbracket \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n \rrbracket$
by simp
moreover have $\llbracket \llbracket ((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \rrbracket_{\text{config}}$
 $= \llbracket \llbracket (K_1 \neg \uparrow n) \# \Gamma \rrbracket_{\text{prim}} \cap \llbracket \llbracket \Psi \rrbracket_{\text{TESL}} \geq n \cap \llbracket \llbracket (K_1 \text{ implies } K_2) \# \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n \rrbracket$
by simp
moreover have $\llbracket \llbracket ((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \rrbracket_{\text{config}}$
 $= \llbracket \llbracket ((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma) \rrbracket_{\text{prim}} \cap \llbracket \llbracket \Psi \rrbracket_{\text{TESL}} \geq n \cap \llbracket \llbracket (K_1 \text{ implies } K_2) \# \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n \rrbracket$
by simp
ultimately show ?thesis
proof –
have $f1: \langle \llbracket K_1 \neg \uparrow n \rrbracket_{\text{prim}} \cup \llbracket K_1 \uparrow n \rrbracket_{\text{prim}} \cap \llbracket K_2 \uparrow n \rrbracket_{\text{prim}} \rrbracket \cap \llbracket K_1 \text{ implies } K_2 \rrbracket_{\text{TESL}} \geq \text{Suc } n \cap (\llbracket \llbracket \Psi \rrbracket_{\text{TESL}} \geq n \cap \llbracket \llbracket \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n \rrbracket) = \llbracket \llbracket (K_1 \text{ implies } K_2) \# \Psi \rrbracket_{\text{TESL}} \geq n \cap \llbracket \llbracket \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n \rrbracket$
using TESL-interp-stepwise-implies-coind-unfold TESL-interpretation-stepwise-cons-morph
by blast
have $\langle \llbracket K_1 \neg \uparrow n \rrbracket_{\text{prim}} \cap \llbracket \llbracket \Gamma \rrbracket_{\text{prim}} \cup \llbracket K_1 \uparrow n \rrbracket_{\text{prim}} \cap \llbracket \llbracket (K_2 \uparrow n) \# \Gamma \rrbracket_{\text{prim}} \rrbracket$

$\llbracket \cdot \rrbracket_{prim} = (\llbracket K_1 \multimap n \rrbracket_{prim} \cup \llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket K_2 \uparrow n \rrbracket_{prim}) \cap \llbracket \Gamma \rrbracket_{prim}$
by force
then have $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi \rrbracket_{config} = (\llbracket K_1 \multimap n \rrbracket_{prim} \cap \llbracket \Gamma \rrbracket_{prim} \cup \llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket (K_2 \uparrow n) \# \Gamma \rrbracket_{prim}) \cap (\llbracket \Psi \rrbracket_{TESL} \geq n \cap \llbracket (K_1 \text{ implies } K_2) \# \Phi \rrbracket_{TESL} \geq \text{Suc } n)$
using f1 by (*simp add: inf-left-commute inf-sup-aci(2)*)
then show *?thesis*
by (*simp add: Int-Un-distrib2 inf-sup-aci(2)*)
qed
qed

lemma *HeronConf-interp-stepwise-implies-not-cases:*

shows $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi \rrbracket_{config}$
 $= \llbracket ((K_1 \multimap n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \rrbracket_{config}$
 $\cup \llbracket ((K_1 \uparrow n) \# (K_2 \multimap n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \rrbracket_{config}$
 \rangle

proof –

have $\langle \llbracket \Gamma, n \vdash (K_1 \text{ implies not } K_2) \# \Psi \triangleright \Phi \rrbracket_{config} = \llbracket \Gamma \rrbracket_{prim} \cap \llbracket (K_1 \text{ implies not } K_2) \# \Psi \rrbracket_{TESL} \geq n \cap \llbracket \Phi \rrbracket_{TESL} \geq \text{Suc } n$

by simp

moreover have $\langle \llbracket ((K_1 \multimap n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \rrbracket_{config}$
 $= \llbracket (K_1 \multimap n) \# \Gamma \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL} \geq n \cap \llbracket (K_1 \text{ implies not } K_2) \# \Phi \rrbracket_{TESL} \geq \text{Suc } n$

by simp

moreover have $\langle \llbracket ((K_1 \uparrow n) \# (K_2 \multimap n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \rrbracket_{config}$
 $= \llbracket ((K_1 \uparrow n) \# (K_2 \multimap n) \# \Gamma) \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL} \geq n \cap \llbracket (K_1 \text{ implies not } K_2) \# \Phi \rrbracket_{TESL} \geq \text{Suc } n$

by simp

ultimately show *?thesis*

proof –

have f1: $\langle (\llbracket K_1 \multimap n \rrbracket_{prim} \cup \llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket K_2 \multimap n \rrbracket_{prim}) \cap \llbracket K_1 \text{ implies not } K_2 \rrbracket_{TESL} \geq \text{Suc } n \cap (\llbracket \Psi \rrbracket_{TESL} \geq n \cap \llbracket \Phi \rrbracket_{TESL} \geq \text{Suc } n)$
 $= \llbracket (K_1 \text{ implies not } K_2) \# \Psi \rrbracket_{TESL} \geq n \cap \llbracket \Phi \rrbracket_{TESL} \geq \text{Suc } n$

using *TESL-interp-stepwise-implies-not-coind-unfold TESL-interpretation-stepwise-cons-morph*
by blast

have $\langle \llbracket K_1 \multimap n \rrbracket_{prim} \cap \llbracket \Gamma \rrbracket_{prim} \cup \llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket (K_2 \multimap n) \# \Gamma \rrbracket_{prim}$
 $\rrbracket_{prim} = (\llbracket K_1 \multimap n \rrbracket_{prim} \cup \llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket K_2 \multimap n \rrbracket_{prim}) \cap \llbracket \Gamma \rrbracket_{prim}$

by force

then have $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi \rrbracket_{config}$
 $= (\llbracket K_1 \multimap n \rrbracket_{prim} \cap \llbracket \Gamma \rrbracket_{prim} \cup \llbracket K_1 \uparrow n \rrbracket_{prim} \cap \llbracket (K_2 \multimap n) \# \Gamma \rrbracket_{prim}) \cap (\llbracket \Psi \rrbracket_{TESL} \geq n \cap \llbracket (K_1 \text{ implies not } K_2) \# \Phi \rrbracket_{TESL} \geq \text{Suc } n)$

using f1 by (*simp add: inf-left-commute inf-sup-aci(2)*)

then show *?thesis*

by (*simp add: Int-Un-distrib2 inf-sup-aci(2)*)

qed

qed

lemma *HeronConf-interp-stepwise-timedelayed-cases:*

shows $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi \rrbracket_{\text{config}}$
 $= \llbracket ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rrbracket_{\text{config}}$
 $\cup \llbracket ((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rrbracket_{\text{config}} \rangle$

proof –

have $1 : \llbracket \Gamma, n \vdash (K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi \triangleright \Phi \rrbracket_{\text{config}}$
 $= \llbracket \Gamma \rrbracket_{\text{prim}} \cap \llbracket (K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi \rrbracket_{\text{TESL}} \geq n \cap \llbracket \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n,$
by *simp*

moreover have $\llbracket ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rrbracket_{\text{config}}$
 $= \llbracket (K_1 \neg\uparrow n) \# \Gamma \rrbracket_{\text{prim}} \cap \llbracket \Psi \rrbracket_{\text{TESL}} \geq n \cap \llbracket (K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n,$
by *simp*

moreover have $\langle \llbracket ((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rrbracket_{\text{config}}$
 $= \llbracket (K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma \rrbracket_{\text{prim}} \cap \llbracket \Psi \rrbracket_{\text{TESL}} \geq n$
 $\cap \llbracket (K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n,$

by *simp*

ultimately show *?thesis*

proof –

have $\langle \llbracket \Gamma, n \vdash (K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi \triangleright \Phi \rrbracket_{\text{config}}$
 $= \llbracket \Gamma \rrbracket_{\text{prim}} \cap (\llbracket (K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi \rrbracket_{\text{TESL}} \geq n$
 $\cap \llbracket \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n) \rangle$

using *1* **by** *blast*

then have $\langle \llbracket \Gamma, n \vdash (K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi \triangleright \Phi \rrbracket_{\text{config}}$
 $= (\llbracket K_1 \neg\uparrow n \rrbracket_{\text{prim}} \cup \llbracket K_1 \uparrow n \rrbracket_{\text{prim}} \cap \llbracket K_2 @ n \oplus \delta\tau \Rightarrow K_3 \rrbracket_{\text{prim}}) \cap$
 $(\llbracket \Gamma \rrbracket_{\text{prim}} \cap (\llbracket \Psi \rrbracket_{\text{TESL}} \geq n \cap \llbracket (K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi \rrbracket_{\text{TESL}} \geq \text{Suc } n)) \rangle$

using *TESL-interpretation-stepwise-cons-morph TESL-interp-stepwise-timedelayed-coind-unfold*

proof –

have $\langle \llbracket (K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi \rrbracket_{\text{TESL}} \geq n =$
 $(\llbracket K_1 \neg\uparrow n \rrbracket_{\text{prim}} \cup \llbracket K_1 \uparrow n \rrbracket_{\text{prim}} \cap \llbracket K_2 @ n \oplus \delta\tau \Rightarrow K_3 \rrbracket_{\text{prim}}) \cap \llbracket K_1$
 $\text{time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3 \rrbracket_{\text{TESL}} \geq \text{Suc } n \cap \llbracket \Psi \rrbracket_{\text{TESL}} \geq n \rangle$

using *TESL-interp-stepwise-timedelayed-coind-unfold TESL-interpretation-stepwise-cons-morph*
by *blast*

then show *?thesis*

by (*simp add: Int-assoc Int-left-commute*)

qed

then show *?thesis* **by** (*simp add: inf-assoc inf-sup-distrib2*)

qed

qed

lemma *HeronConf-interp-stepwise-weakly-precedes-cases:*

shows $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi \rrbracket_{\text{config}}$

$= \llbracket ((\lceil \#^{\leq} K_2 n, \#^{\leq} K_1 n \rceil \in (\lambda(x,y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \rrbracket_{config}$

proof –

have $\langle \llbracket \Gamma, n \vdash (K_1 \text{ weakly precedes } K_2) \# \Psi \triangleright \Phi \rrbracket_{config} = \llbracket \llbracket \Gamma \rrbracket_{prim} \cap \llbracket (K_1 \text{ weakly precedes } K_2) \# \Psi \rrbracket_{TESL}^{\geq n} \cap \llbracket \llbracket \Phi \rrbracket_{TESL}^{\geq Suc\ n} \rrbracket$

by simp

moreover have $\langle \llbracket ((\lceil \#^{\leq} K_2 n, \#^{\leq} K_1 n \rceil \in (\lambda(x,y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \rrbracket_{config}$

$= \llbracket \llbracket (\lceil \#^{\leq} K_2 n, \#^{\leq} K_1 n \rceil \in (\lambda(x,y). x \leq y)) \# \Gamma \rrbracket_{prim} \cap \llbracket \llbracket \Psi \rrbracket_{TESL}^{\geq n} \cap \llbracket \llbracket (K_1 \text{ weakly precedes } K_2) \# \Phi \rrbracket_{TESL}^{\geq Suc\ n} \rrbracket$

by simp

ultimately show ?thesis

proof –

have $\langle \llbracket \lceil \#^{\leq} K_2 n, \#^{\leq} K_1 n \rceil \in (\lambda(x,y). x \leq y) \rrbracket_{prim} \cap \llbracket K_1 \text{ weakly precedes } K_2 \rrbracket_{TESL}^{\geq Suc\ n} \cap \llbracket \llbracket \Psi \rrbracket_{TESL}^{\geq n} = \llbracket \llbracket (K_1 \text{ weakly precedes } K_2) \# \Psi \rrbracket_{TESL}^{\geq n} \rrbracket$

using *TESL-interp-stepwise-weakly-precedes-coind-unfold TESL-interpretation-stepwise-cons-morph*

by blast

then show ?thesis

by auto

qed

qed

lemma *HeronConf-interp-stepwise-strictly-precedes-cases:*

shows $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi \rrbracket_{config}$

$= \llbracket ((\lceil \#^{\leq} K_2 n, \#^{<} K_1 n \rceil \in (\lambda(x,y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi) \rrbracket_{config}$

proof –

have $\langle \llbracket \Gamma, n \vdash (K_1 \text{ strictly precedes } K_2) \# \Psi \triangleright \Phi \rrbracket_{config} = \llbracket \llbracket \Gamma \rrbracket_{prim} \cap \llbracket (K_1 \text{ strictly precedes } K_2) \# \Psi \rrbracket_{TESL}^{\geq n} \cap \llbracket \llbracket \Phi \rrbracket_{TESL}^{\geq Suc\ n} \rrbracket$

by simp

moreover have $\langle \llbracket ((\lceil \#^{\leq} K_2 n, \#^{<} K_1 n \rceil \in (\lambda(x,y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi) \rrbracket_{config}$

$= \llbracket \llbracket (\lceil \#^{\leq} K_2 n, \#^{<} K_1 n \rceil \in (\lambda(x,y). x \leq y)) \# \Gamma \rrbracket_{prim} \cap \llbracket \llbracket \Psi \rrbracket_{TESL}^{\geq n} \cap \llbracket \llbracket (K_1 \text{ strictly precedes } K_2) \# \Phi \rrbracket_{TESL}^{\geq Suc\ n} \rrbracket$

by simp

ultimately show ?thesis

proof –

have $\langle \llbracket \lceil \#^{\leq} K_2 n, \#^{<} K_1 n \rceil \in (\lambda(x,y). x \leq y) \rrbracket_{prim} \cap \llbracket K_1 \text{ strictly precedes } K_2 \rrbracket_{TESL}^{\geq Suc\ n} \cap \llbracket \llbracket \Psi \rrbracket_{TESL}^{\geq n} = \llbracket \llbracket (K_1 \text{ strictly precedes } K_2) \# \Psi \rrbracket_{TESL}^{\geq n} \rrbracket$

using *TESL-interp-stepwise-strictly-precedes-coind-unfold TESL-interpretation-stepwise-cons-morph*

by blast

then show ?thesis

by auto

qed

qed

lemma *HeronConf-interp-stepwise-kills-cases:*

shows $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi \rrbracket_{config}$

$$\begin{aligned}
&= \llbracket ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rrbracket_{config} \\
&\cup \llbracket ((K_1 \uparrow n) \# (K_2 \neg\uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rrbracket_{config} \\
\text{proof} - \\
&\quad \text{have } \langle \llbracket \Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi \rrbracket_{config} = \llbracket \Gamma \rrbracket_{prim} \cap \llbracket (K_1 \text{ kills } K_2) \# \Psi \rrbracket_{TESL}^{\geq n} \cap \llbracket \Phi \rrbracket_{TESL}^{\geq Suc\ n} \\
&\quad \text{by } simp \\
&\quad \text{moreover have } \llbracket ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rrbracket_{config} \\
&\quad = \llbracket (K_1 \neg\uparrow n) \# \Gamma \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL}^{\geq n} \cap \llbracket (K_1 \text{ kills } K_2) \# \Phi \rrbracket_{TESL}^{\geq Suc\ n} \\
&\quad \text{by } simp \\
&\quad \text{moreover have } \langle \llbracket (K_1 \uparrow n) \# (K_2 \neg\uparrow \geq n) \# \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rrbracket_{config} \\
&\quad = \llbracket (K_1 \uparrow n) \# (K_2 \neg\uparrow \geq n) \# \Gamma \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL}^{\geq n} \cap \llbracket (K_1 \text{ kills } K_2) \# \Phi \rrbracket_{TESL}^{\geq Suc\ n} \\
&\quad \text{by } simp \\
&\quad \text{ultimately show } ?thesis \\
&\quad \text{proof} - \\
&\quad \quad \text{have } \langle \llbracket (K_1 \text{ kills } K_2) \# \Psi \rrbracket_{TESL}^{\geq n} = (\llbracket (K_1 \neg\uparrow n) \rrbracket_{prim} \cup \llbracket (K_1 \uparrow n) \rrbracket_{prim} \cap \llbracket (K_2 \neg\uparrow \geq n) \rrbracket_{prim}) \cap \llbracket (K_1 \text{ kills } K_2) \rrbracket_{TESL}^{\geq Suc\ n} \cap \llbracket \Psi \rrbracket_{TESL}^{\geq n} \\
&\quad \quad \text{using } TESL\text{-interp-stepwise-kills-coind-unfold } TESL\text{-interpretation-stepwise-cons-morph} \\
&\quad \quad \text{by } blast \\
&\quad \quad \text{then show } ?thesis \\
&\quad \quad \text{by } auto \\
&\quad \text{qed} \\
&\text{qed} \\
&\text{end}
\end{aligned}$$

Chapter 6

Main Theorems

```
theory Hygge-Theory
imports
  Corecursive-Prop
```

```
begin
```

6.1 Initial configuration

Solving a specification Ψ means to start operational semantics at initial configuration \square , $0 \vdash \Psi \triangleright \square$

theorem *solve-start*:

```
shows  $\langle \llbracket \Psi \rrbracket_{TESL} = \llbracket \square, 0 \vdash \Psi \triangleright \square \rrbracket_{config} \rangle$ 
proof -
  have  $\langle \llbracket \Psi \rrbracket_{TESL} = \llbracket \Psi \rrbracket_{TESL}^{\geq 0} \rangle$ 
  by (simp add: TESL-interpretation-stepwise-zero)
  moreover have  $\langle \llbracket \square, 0 \vdash \Psi \triangleright \square \rrbracket_{config} = \llbracket \square \rrbracket_{prim} \cap \llbracket \Psi \rrbracket_{TESL}^{\geq 0} \cap \llbracket \square \rrbracket_{TESL}^{\geq Suc\ 0} \rangle$ 
  by simp
  ultimately show ?thesis by auto
qed
```

6.2 Soundness

lemma *sound-reduction*:

```
assumes  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle \hookrightarrow_i \langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle$ 
shows  $\langle \llbracket \Gamma_1 \rrbracket_{prim} \cap \llbracket \Psi_1 \rrbracket_{TESL}^{\geq n_1} \cap \llbracket \Phi_1 \rrbracket_{TESL}^{\geq Suc\ n_1} \rangle$ 
 $\supseteq \llbracket \Gamma_2 \rrbracket_{prim} \cap \llbracket \Psi_2 \rrbracket_{TESL}^{\geq n_2} \cap \llbracket \Phi_2 \rrbracket_{TESL}^{\geq Suc\ n_2} \rangle$  (is ?P)
proof -
  from assms consider
    (a)  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle \hookrightarrow_i \langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle$ 
  | (b)  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle \hookrightarrow_e \langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle$ 
  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  $\Gamma$   $n$   $K_1$   $\tau$   $K_2$   $\Psi$   $\Phi$ 
      assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash (K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi \triangleright \Phi)$ 
      and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (\Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi))$ 
      thus ?P
      using HeronConf-interp-stepwise-sporadicon-cases HeronConf-interpretation.simps
    by blast
  next
    fix  $\Gamma$   $n$   $K_1$   $\tau$   $K_2$   $\Psi$   $\Phi$ 
    assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash (K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi \triangleright \Phi)$ 
    and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi)$ 
    thus ?P
    using HeronConf-interp-stepwise-sporadicon-cases HeronConf-interpretation.simps
  by blast
  next
    fix  $\Gamma$   $n$   $K_1$   $K_2$   $R$   $\Psi$   $\Phi$ 
    assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash (\text{time-relation } [K_1, K_2] \in R) \# \Psi$ 
     $\triangleright \Phi)$ 
    and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (((\tau_{var} (K_1, n), \tau_{var} (K_2, n)) \in R) \# \Gamma), n \vdash$ 
     $\Psi \triangleright ((\text{time-relation } [K_1, K_2] \in R) \# \Phi))$ 
    thus ?P
    using HeronConf-interp-stepwise-tagrel-cases HeronConf-interpretation.simps
  by blast
  next
    fix  $\Gamma$   $n$   $K_1$   $K_2$   $\Psi$   $\Phi$ 
    assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \triangleright \Phi)$ 
    and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \#$ 
     $\Phi))$ 
    thus ?P
    using HeronConf-interp-stepwise-implies-cases HeronConf-interpretation.simps
  by blast
  next
    fix  $\Gamma$   $n$   $K_1$   $K_2$   $\Psi$   $\Phi$ 
    assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi)$ 
    and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1$ 
     $\text{ implies } K_2) \# \Phi))$ 
    thus ?P
    using HeronConf-interp-stepwise-implies-cases HeronConf-interpretation.simps
  by blast
  next
    fix  $\Gamma$   $n$   $K_1$   $K_2$   $\Psi$   $\Phi$ 
    assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi)$ 
    and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2)$ 

```

```

#  $\Phi$ ) $\rangle$ 
  thus ?P
  using HeronConf-interp-stepwise-implies-not-cases HeronConf-interpretation.simps
by blast
next
  fix  $\Gamma$   $n$   $K_1$   $K_2$   $\Psi$   $\Phi$ 
  assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi)$ 
  and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (((K_1 \uparrow n) \# (K_2 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1$ 
implies not  $K_2) \# \Phi))$ 
  thus ?P
  using HeronConf-interp-stepwise-implies-not-cases HeronConf-interpretation.simps
by blast
next
  fix  $\Gamma$   $n$   $K_1$   $\delta\tau$   $K_2$   $K_3$   $\Psi$   $\Phi$ 
  assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2$ 
implies  $K_3) \# \Psi) \triangleright \Phi)$ 
  and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed}$ 
by  $\delta\tau$  on  $K_2$  implies  $K_3) \# \Phi))$ 
  thus ?P
  using HeronConf-interp-stepwise-timedelayed-cases HeronConf-interpretation.simps
by blast
next
  fix  $\Gamma$   $n$   $K_1$   $\delta\tau$   $K_2$   $K_3$   $\Psi$   $\Phi$ 
  assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2$ 
implies  $K_3) \# \Psi) \triangleright \Phi)$ 
  and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n \vdash$ 
 $\Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi))$ 
  thus ?P
  using HeronConf-interp-stepwise-timedelayed-cases HeronConf-interpretation.simps
by blast
next
  fix  $\Gamma$   $n$   $K_1$   $K_2$   $\Psi$   $\Phi$ 
  assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi)$ 
  and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (((\lceil \# \leq K_2 n, \# \leq K_1 n \rceil \in (\lambda(x, y). x \leq y)) \#$ 
 $\Gamma), n \vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi))$ 
  thus ?P
  using HeronConf-interp-stepwise-weakly-precedes-cases HeronConf-interpretation.simps
by blast
next
  fix  $\Gamma$   $n$   $K_1$   $K_2$   $\Psi$   $\Phi$ 
  assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi)$ 
  and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (((\lceil \# \leq K_2 n, \# < K_1 n \rceil \in (\lambda(x, y). x \leq y)) \#$ 
 $\Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi))$ 
  thus ?P
  using HeronConf-interp-stepwise-strictly-precedes-cases HeronConf-interpretation.simps
by blast
next
  fix  $\Gamma$   $n$   $K_1$   $K_2$   $\Psi$   $\Phi$ 
  assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi)$ 

```

```

    and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (\langle (K_1 \neg \uparrow n) \# \Gamma \rangle, n \vdash \Psi \triangleright \langle (K_1 \text{ kills } K_2) \# \Phi \rangle)$ 
    thus ?P
    using HeronConf-interp-stepwise-kills-cases HeronConf-interpretation.simps
  by blast
  next
    fix  $\Gamma \ n \ K_1 \ K_2 \ \Psi \ \Phi$ 
    assume  $\langle \Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1 \rangle = (\Gamma, n \vdash \langle (K_1 \text{ kills } K_2) \# \Psi \rangle \triangleright \Phi)$ 
    and  $\langle \Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2 \rangle = (\langle (K_1 \uparrow n) \# (K_2 \neg \uparrow \geq n) \# \Gamma \rangle, n \vdash \Psi \triangleright \langle (K_1 \text{ kills } K_2) \# \Phi \rangle)$ 
    thus ?P
    using HeronConf-interp-stepwise-kills-cases HeronConf-interpretation.simps
  by blast
  qed
  qed
  qed

```

inductive-cases *step-elim*: $\langle \mathcal{S}_1 \hookrightarrow \mathcal{S}_2 \rangle$

```

lemma sound-reduction':
  assumes  $\langle \mathcal{S}_1 \hookrightarrow \mathcal{S}_2 \rangle$ 
  shows  $\langle \llbracket \mathcal{S}_1 \rrbracket_{\text{config}} \supseteq \llbracket \mathcal{S}_2 \rrbracket_{\text{config}} \rangle$ 
proof –
  have  $\langle \forall s_1 \ s_2. (\llbracket s_2 \rrbracket_{\text{config}} \subseteq \llbracket s_1 \rrbracket_{\text{config}}) \vee \neg(s_1 \hookrightarrow s_2) \rangle$ 
    using sound-reduction by fastforce
  thus ?thesis using assms by blast
qed

```

```

lemma sound-reduction-generalized:
  assumes  $\langle \mathcal{S}_1 \hookrightarrow^k \mathcal{S}_2 \rangle$ 
  shows  $\langle \llbracket \mathcal{S}_1 \rrbracket_{\text{config}} \supseteq \llbracket \mathcal{S}_2 \rrbracket_{\text{config}} \rangle$ 
proof –
  from assms show ?thesis
  proof (induct k arbitrary:  $\mathcal{S}_2$ )
    case 0
    hence *:  $\langle \mathcal{S}_1 \hookrightarrow^0 \mathcal{S}_2 \implies \mathcal{S}_1 = \mathcal{S}_2 \rangle$  by auto
    moreover have  $\langle \mathcal{S}_1 = \mathcal{S}_2 \rangle$  using * 0.prems by linarith
    ultimately show ?case by auto
  next
    case (Suc k)
    thus ?case
    proof –
      fix  $k :: \text{nat}$ 
      assume ff:  $\langle \mathcal{S}_1 \hookrightarrow^{\text{Suc } k} \mathcal{S}_2 \rangle$ 
      assume hi:  $\langle \bigwedge \mathcal{S}_2. \mathcal{S}_1 \hookrightarrow^k \mathcal{S}_2 \implies \llbracket \mathcal{S}_2 \rrbracket_{\text{config}} \subseteq \llbracket \mathcal{S}_1 \rrbracket_{\text{config}} \rangle$ 
      obtain  $\mathcal{S}_n$  where red-decomp:  $\langle \mathcal{S}_1 \hookrightarrow^k \mathcal{S}_n \rangle \wedge \langle \mathcal{S}_n \hookrightarrow \mathcal{S}_2 \rangle$  using ff by
auto
      hence  $\langle \llbracket \mathcal{S}_1 \rrbracket_{\text{config}} \supseteq \llbracket \mathcal{S}_n \rrbracket_{\text{config}} \rangle$  using hi by simp
      also have  $\langle \llbracket \mathcal{S}_n \rrbracket_{\text{config}} \supseteq \llbracket \mathcal{S}_2 \rrbracket_{\text{config}} \rangle$  by (simp add: red-decomp
sound-reduction')
    qed
  qed

```

ultimately show $\langle \llbracket \mathcal{S}_1 \rrbracket_{config} \supseteq \llbracket \mathcal{S}_2 \rrbracket_{config} \rangle$ by *simp*
 qed
 qed
 qed

From initial configuration, any reduction step number k providing a configuration \mathcal{S} will denote runs from initial specification Ψ .

theorem *soundness*:

assumes $\langle \langle \llbracket \cdot \rrbracket, \emptyset \vdash \Psi \triangleright \llbracket \cdot \rrbracket \rangle \hookrightarrow^k \mathcal{S} \rangle$
 shows $\langle \llbracket \llbracket \Psi \rrbracket_{TESL} \supseteq \llbracket \mathcal{S} \rrbracket_{config} \rangle$
 using *assms sound-reduction-generalized solve-start* by *blast*

6.3 Completeness

lemma *complete-direct-successors*:

shows $\langle \llbracket \Gamma, n \vdash \Psi \triangleright \Phi \rrbracket_{config} \subseteq (\bigcup_{X \in \mathcal{C}_{next}} \langle \Gamma, n \vdash \Psi \triangleright \Phi \rangle. \llbracket X \rrbracket_{config}) \rangle$
 proof (induct Ψ)
 case *Nil*
 show ?case
 using *HeronConf-interp-stepwise-instant-cases operational-semantic-step.simps*
operational-semantic-intro.instant-i
 by *fastforce*
 next
 case (*Cons* $\psi \Psi$)
 then show ?case
 proof (cases ψ)
 case (*SporadicOn* $K1 \tau K2$)
 then show ?thesis
 using *HeronConf-interp-stepwise-sporadicon-cases*[of $\langle \Gamma \rangle \langle n \rangle \langle K1 \rangle \langle \tau \rangle \langle K2 \rangle$
 $\langle \Psi \rangle \langle \Phi \rangle$]
Cnext-solve-sporadicon[of $\langle \Gamma \rangle \langle n \rangle \langle \Psi \rangle \langle K1 \rangle \langle \tau \rangle \langle K2 \rangle \langle \Phi \rangle$] by *blast*
 next
 case (*TagRelation* $K_1 K_2 R$)
 then show ?thesis
 using *HeronConf-interp-stepwise-tagrel-cases*[of $\langle \Gamma \rangle \langle n \rangle \langle K_1 \rangle \langle K_2 \rangle \langle R \rangle \langle \Psi \rangle$
 $\langle \Phi \rangle$]
Cnext-solve-tagrel[of $\langle K_1 \rangle \langle n \rangle \langle K_2 \rangle \langle R \rangle \langle \Gamma \rangle \langle \Psi \rangle \langle \Phi \rangle$] by *blast*
 next
 case (*Implies* $K1 K2$)
 then show ?thesis
 using *HeronConf-interp-stepwise-implies-cases*[of $\langle \Gamma \rangle \langle n \rangle \langle K1 \rangle \langle K2 \rangle \langle \Psi \rangle$
 $\langle \Phi \rangle$]
Cnext-solve-implies[of $\langle K1 \rangle \langle n \rangle \langle \Gamma \rangle \langle \Psi \rangle \langle K2 \rangle \langle \Phi \rangle$] by *blast*
 next
 case (*ImpliesNot* $K1 K2$)
 then show ?thesis
 using *HeronConf-interp-stepwise-implies-not-cases*[of $\langle \Gamma \rangle \langle n \rangle \langle K1 \rangle \langle K2 \rangle$
 $\langle \Psi \rangle \langle \Phi \rangle$]
Cnext-solve-implies-not[of $\langle K1 \rangle \langle n \rangle \langle \Gamma \rangle \langle \Psi \rangle \langle K2 \rangle \langle \Phi \rangle$] by *blast*

```

next
  case (TimeDelayedBy Kmast  $\tau$  Kmeas Kslave)
  thus ?thesis
    using HeronConf-interp-stepwise-timedelayed-cases[of  $\langle \Gamma \rangle \langle n \rangle \langle K_{\text{mast}} \rangle \langle \tau \rangle$ 
 $\langle K_{\text{meas}} \rangle \langle K_{\text{slave}} \rangle \langle \Psi \rangle \langle \Phi \rangle$ ]
      Cnext-solve-timedelayed[of  $\langle K_{\text{mast}} \rangle \langle n \rangle \langle \Gamma \rangle \langle \Psi \rangle \langle \tau \rangle \langle K_{\text{meas}} \rangle \langle K_{\text{slave}} \rangle$ 
 $\langle \Phi \rangle$ ] by blast
    next
      case (WeaklyPrecedes K1 K2)
      then show ?thesis
        using HeronConf-interp-stepwise-weakly-precedes-cases[of  $\langle \Gamma \rangle \langle n \rangle \langle K_1 \rangle$ 
 $\langle K_2 \rangle \langle \Psi \rangle \langle \Phi \rangle$ ]
          Cnext-solve-weakly-precedes[of  $\langle K_2 \rangle \langle n \rangle \langle K_1 \rangle \langle \Gamma \rangle \langle \Psi \rangle \langle \Phi \rangle$ ]
          by blast
        next
          case (StrictlyPrecedes K1 K2)
          then show ?thesis
            using HeronConf-interp-stepwise-strictly-precedes-cases[of  $\langle \Gamma \rangle \langle n \rangle \langle K_1 \rangle$ 
 $\langle K_2 \rangle \langle \Psi \rangle \langle \Phi \rangle$ ]
              Cnext-solve-strictly-precedes[of  $\langle K_2 \rangle \langle n \rangle \langle K_1 \rangle \langle \Gamma \rangle \langle \Psi \rangle \langle \Phi \rangle$ ]
              by blast
            next
              case (Kills K1 K2)
              then show ?thesis
                using HeronConf-interp-stepwise-kills-cases[of  $\langle \Gamma \rangle \langle n \rangle \langle K_1 \rangle \langle K_2 \rangle \langle \Psi \rangle \langle \Phi \rangle$ ]
                  Cnext-solve-kills[of  $\langle K_1 \rangle \langle n \rangle \langle \Gamma \rangle \langle \Psi \rangle \langle K_2 \rangle \langle \Phi \rangle$ ] by blast
                qed
              qed
            qed
          qed
        qed
      qed
    qed
  qed

```

lemma *complete-direct-successors'*:
shows $\langle \llbracket \mathcal{S} \rrbracket_{\text{config}} \subseteq (\bigcup X \in \mathcal{C}_{\text{next}} \mathcal{S}. \llbracket X \rrbracket_{\text{config}}) \rangle$
proof –
from *HeronConf-interpretation.cases* **obtain** $\Gamma \ n \ \Psi \ \Phi$ **where** $\langle \mathcal{S} = (\Gamma, n \vdash \Psi \triangleright \Phi) \rangle$ **by** *blast*
with *complete-direct-successors*[*of* $\langle \Gamma \rangle \langle n \rangle \langle \Psi \rangle \langle \Phi \rangle$] **show** ?thesis **by** *simp*
qed

lemma *branch-existence*:
assumes $\langle \varrho \in \llbracket \mathcal{S}_1 \rrbracket_{\text{config}} \rangle$
shows $\langle \exists \mathcal{S}_2. (\mathcal{S}_1 \hookrightarrow \mathcal{S}_2) \wedge (\varrho \in \llbracket \mathcal{S}_2 \rrbracket_{\text{config}}) \rangle$
proof –
from *assms complete-direct-successors'* **have** $\langle \varrho \in (\bigcup X \in \mathcal{C}_{\text{next}} \mathcal{S}_1. \llbracket X \rrbracket_{\text{config}}) \rangle$
by *blast*
hence $\langle \exists s \in \mathcal{C}_{\text{next}} \mathcal{S}_1. \varrho \in \llbracket s \rrbracket_{\text{config}} \rangle$ **by** *simp*
thus ?thesis **by** *blast*
qed

lemma *branch-existence'*:
assumes $\langle \varrho \in \llbracket \mathcal{S}_1 \rrbracket_{\text{config}} \rangle$


```

  shows  $\langle \exists \mathcal{S}_2. (\mathcal{S}_1 \hookrightarrow^k \mathcal{S}_2) \wedge (\varrho \in \llbracket \mathcal{S}_2 \rrbracket_{config}) \rangle$ 
proof (induct k)
  case 0
    then show ?case by (simp add: assms)
  next
    case (Suc k)
      then show ?case
        using branch-existence relpoup-Suc-I[of  $\langle k \rangle$   $\langle operational-semantic-step \rangle$ ] by
blast
qed

```

Any run from initial specification Ψ has a corresponding configuration \mathcal{S} at any reduction step number k starting from initial configuration.

```

theorem completeness:
  assumes  $\langle \varrho \in \llbracket \llbracket \Psi \rrbracket_{TESL} \rrbracket$ 
  shows  $\langle \exists \mathcal{S}. (\llbracket \cdot \rrbracket, 0 \vdash \Psi \triangleright \llbracket \cdot \rrbracket) \hookrightarrow^k \mathcal{S} \rangle$ 
     $\wedge \varrho \in \llbracket \mathcal{S} \rrbracket_{config}$ 
  using assms branch-existence' solve-start by blast

```

6.4 Progress

```

lemma instant-index-increase:
  assumes  $\langle \varrho \in \llbracket \Gamma, n \vdash \Psi \triangleright \Phi \rrbracket_{config} \rangle$ 
  shows  $\langle \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma, n \vdash \Psi \triangleright \Phi) \hookrightarrow^k (\Gamma_k, Suc\ n \vdash \Psi_k \triangleright \Phi_k)) \rangle$ 
     $\wedge \varrho \in \llbracket \Gamma_k, Suc\ n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config}$ 
proof (insert assms, induct  $\Psi$  arbitrary:  $\Gamma\ \Phi$ )
  case (Nil  $\Gamma\ \Phi$ )
    then show ?case
      proof -
        have  $\langle (\Gamma, n \vdash \llbracket \cdot \rrbracket \triangleright \Phi) \hookrightarrow^1 (\Gamma, Suc\ n \vdash \Phi \triangleright \llbracket \cdot \rrbracket) \rangle$ 
          using instant-i intro-part by fastforce
        moreover have  $\langle \llbracket \Gamma, n \vdash \llbracket \cdot \rrbracket \triangleright \Phi \rrbracket_{config} = \llbracket \Gamma, Suc\ n \vdash \Phi \triangleright \llbracket \cdot \rrbracket \rrbracket_{config} \rangle$ 
          by auto
        moreover have  $\langle \varrho \in \llbracket \Gamma, Suc\ n \vdash \Phi \triangleright \llbracket \cdot \rrbracket \rrbracket_{config} \rangle$ 
          using assms Nil.premis calculation(2) by blast
        ultimately show ?thesis by blast
      qed
    qed
  next
    case (Cons  $\psi\ \Psi$ )
      then show ?case
        proof (induct  $\psi$ )
          case (SporadicOn  $K_1\ \tau\ K_2$ )
            have branches:  $\langle \llbracket \Gamma, n \vdash ((K_1\ sporadic\ \tau\ on\ K_2) \# \Psi) \triangleright \Phi \rrbracket_{config} \rangle$ 
               $= \llbracket \Gamma, n \vdash \Psi \triangleright ((K_1\ sporadic\ \tau\ on\ K_2) \# \Phi) \rrbracket_{config}$ 
               $\cup \llbracket ((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi \rrbracket_{config}$ 
            using HeronConf-interp-stepwise-sporadicon-cases by simp
            have br1:  $\langle \varrho \in \llbracket \Gamma, n \vdash \Psi \triangleright ((K_1\ sporadic\ \tau\ on\ K_2) \# \Phi) \rrbracket_{config} \rangle$ 
               $\implies \exists \Gamma_k \Psi_k \Phi_k k.$ 

```

$(\langle \Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi \rangle \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
 $\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}}$
proof –
assume $h1: \langle \varrho \in \llbracket \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
hence $\langle \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi)) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \wedge (\varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}}) \rangle$
using $h1 \text{ SporadicOn.premis}$ **by** *simp*
from this obtain $\Gamma_k \Psi_k \Phi_k k$ **where**
 $fp: \langle (\Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi)) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
 $\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}}$ **by** *blast*
have
 $\langle (\Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow (\Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi)) \rangle$
by (*simp add: elims-part sporadic-on-e1*)
with *fp relpowp-Suc-I2* **have**
 $\langle (\Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$ **by** *auto*
thus *?thesis* **using** *fp* **by** *blast*
qed
have $br2: \langle \varrho \in \llbracket ((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi \rrbracket_{\text{config}} \implies \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$
proof –
assume $h2: \langle \varrho \in \llbracket ((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi \rrbracket_{\text{config}} \rangle$
hence $\langle \exists \Gamma_k \Psi_k \Phi_k k. (((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$
using $h2 \text{ SporadicOn.premis}$ **by** *simp*
from this obtain $\Gamma_k \Psi_k \Phi_k k$ **where** $fp: \langle (((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
and $rc: \langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$ **by** *blast*
have $pc: \langle (\Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow (((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi) \rangle$ **by** (*simp add: elims-part sporadic-on-e2*)
hence $\langle (\Gamma, n \vdash ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
using *fp relpowp-Suc-I2* **by** *auto*
with *rc* **show** *?thesis* **by** *blast*
qed
from branches SporadicOn.premis(2) have
 $\langle \varrho \in \llbracket \Gamma, n \vdash \Psi \triangleright ((K_1 \text{ sporadic } \tau \text{ on } K_2) \# \Phi) \rrbracket_{\text{config}} \cup \llbracket ((K_1 \uparrow n) \# (K_2 \downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi \rrbracket_{\text{config}} \rangle$
by *simp*

```

with br1 br2 show ?case by blast
next
case (TagRelation K1 K2 R)
have branches: ⟨[Γ, n ⊢ ((time-relation [K1, K2] ∈ R) # Ψ) ▷ Φ] config
= [([τvar(K1, n), τvar(K2, n)] ∈ R) # Γ), n
⊢ Ψ ▷ ((time-relation [K1, K2] ∈ R) # Φ)] config⟩
using HeronConf-interp-stepwise-tagrel-cases by simp
thus ?case
proof -
have ⟨∃ Γk Ψk Φk k.
((( [τvar(K1, n), τvar(K2, n)] ∈ R) # Γ), n ⊢ Ψ ▷ ((time-relation
[K1, K2] ∈ R) # Φ))
↪k (Γk, Suc n ⊢ Ψk ▷ Φk⟩)⟩ ∧ ρ ∈ [Γk, Suc n ⊢ Ψk ▷ Φk] config⟩
using TagRelation.premis by simp

from this obtain Γk Ψk Φk k
where fp:⟨((( [τvar(K1, n), τvar(K2, n)] ∈ R) # Γ), n
⊢ Ψ ▷ ((time-relation [K1, K2] ∈ R) # Φ))
↪k (Γk, Suc n ⊢ Ψk ▷ Φk⟩)⟩
and rc:⟨ρ ∈ [Γk, Suc n ⊢ Ψk ▷ Φk] config⟩ by blast
have pc:⟨(Γ, n ⊢ ((time-relation [K1, K2] ∈ R) # Ψ) ▷ Φ)
↪ ((( [τvar(K1, n), τvar(K2, n)] ∈ R) # Γ), n
⊢ Ψ ▷ ((time-relation [K1, K2] ∈ R) # Φ))⟩
by (simp add: elim-part tagrel-e)
hence ⟨(Γ, n ⊢ (time-relation [K1, K2] ∈ R) # Ψ ▷ Φ) ↪Suc k (Γk, Suc
n ⊢ Ψk ▷ Φk⟩)⟩
using fp relpoup-Suc-I2 by auto
with rc show ?thesis by blast
qed
next
case (Implies K1 K2)
have branches: ⟨[Γ, n ⊢ ((K1 implies K2) # Ψ) ▷ Φ] config
= [((K1 ↗ n) # Γ), n ⊢ Ψ ▷ ((K1 implies K2) # Φ)] config
∪ [((K1 ↗ n) # (K2 ↗ n) # Γ), n ⊢ Ψ ▷ ((K1 implies K2) # Φ)] config⟩
using HeronConf-interp-stepwise-implies-cases by simp
moreover have br1: ⟨ρ ∈ [((K1 ↗ n) # Γ), n ⊢ Ψ ▷ ((K1 implies K2) #
Φ)] config
⇒ ∃ Γk Ψk Φk k. ((Γ, n ⊢ ((K1 implies K2) # Ψ) ▷ Φ)
↪k (Γk, Suc n ⊢ Ψk ▷ Φk⟩)
∧ ρ ∈ [Γk, Suc n ⊢ Ψk ▷ Φk] config⟩
proof -
assume h1: ⟨ρ ∈ [((K1 ↗ n) # Γ), n ⊢ Ψ ▷ ((K1 implies K2) # Φ)
] config⟩
then have ⟨∃ Γk Ψk Φk k.
((( (K1 ↗ n) # Γ), n ⊢ Ψ ▷ ((K1 implies K2) # Φ)) ↪k (Γk,
Suc n ⊢ Ψk ▷ Φk⟩)
∧ ρ ∈ [Γk, Suc n ⊢ Ψk ▷ Φk] config⟩
using h1 Implies.premis by simp
from this obtain Γk Ψk Φk k where

```

$fp: \langle (((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi)) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
and $rc: \langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$ **by** *blast*
have $pc: \langle (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \triangleright \Phi) \hookrightarrow (((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi)) \rangle$
by (*simp add: elims-part implies-e1*)
hence $\langle (\Gamma, n \vdash (K_1 \text{ implies } K_2) \# \Psi \triangleright \Phi) \hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
using *fp relpowp-Suc-I2* **by** *auto*
with *rc* **show** *?thesis* **by** *blast*
qed
moreover have $br2: \langle \varrho \in \llbracket ((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
 $\implies \exists \Gamma_k \Psi_k \Phi_k k. \langle (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
 $\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$
proof –
assume $h2: \langle \varrho \in \llbracket ((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
then have $\langle \exists \Gamma_k \Psi_k \Phi_k k. \langle (((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi)) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle \wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$
using *h2 Implies.premis* **by** *simp*
from this obtain $\Gamma_k \Psi_k \Phi_k k$ **where**
 $fp: \langle (((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi)) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
and $rc: \langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$ **by** *blast*
have $\langle (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow (((K_1 \uparrow n) \# (K_2 \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies } K_2) \# \Phi)) \rangle$
by (*simp add: elims-part implies-e2*)
hence $\langle (\Gamma, n \vdash ((K_1 \text{ implies } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
using *fp relpowp-Suc-I2* **by** *auto*
with *rc* **show** *?thesis* **by** *blast*
qed
ultimately show *?case* **using** *Implies.premis(2)* **by** *blast*
next
case (*ImpliesNot* $K_1 K_2$)
have $\text{branches}: \langle \llbracket \Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi \rrbracket_{\text{config}} = \llbracket ((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \rrbracket_{\text{config}} \cup \llbracket ((K_1 \uparrow n) \# (K_2 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
using *HeronConf-interp-stepwise-implies-not-cases* **by** *simp*
moreover have $br1: \langle \varrho \in \llbracket ((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
 $\implies \exists \Gamma_k \Psi_k \Phi_k k. \langle (\Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
 $\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$
proof –

assume $h1: \langle \varrho \in \llbracket ((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
then have $\langle \exists \Gamma_k \Psi_k \Phi_k k.$
 $((((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi)) \hookrightarrow^k (\Gamma_k,$
 $\text{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \rangle$
 $\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$
using $h1 \text{ ImpliesNot.prem } \text{by simp}$
from this obtain $\Gamma_k \Psi_k \Phi_k k$ **where**
 $fp: \langle (((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi)) \hookrightarrow^k (\Gamma_k, \text{Suc }$
 $n \vdash \Psi_k \triangleright \Phi_k) \rangle$
and $rc: \langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$ **by blast**
have $pc: \langle (\Gamma, n \vdash (K_1 \text{ implies not } K_2) \# \Psi \triangleright \Phi)$
 $\hookrightarrow (((K_1 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi)) \rangle$
by ($\text{simp add: elims-part implies-not-e1}$)
hence $\langle (\Gamma, n \vdash (K_1 \text{ implies not } K_2) \# \Psi \triangleright \Phi) \hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright$
 $\Phi_k) \rangle$
using $fp \text{ relpowp-Suc-I2}$ **by auto**
with rc show $?thesis$ **by blast**
qed
moreover have $br2: \langle \varrho \in \llbracket ((K_1 \uparrow n) \# (K_2 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1$
 $\text{implies not } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
 $\implies \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi)$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
 $\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$
proof –
assume $h2: \langle \varrho \in \llbracket ((K_1 \uparrow n) \# (K_2 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies$
 $\text{not } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
then have $\langle \exists \Gamma_k \Psi_k \Phi_k k. ($
 $((K_1 \uparrow n) \# (K_2 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2)$
 $\# \Phi) \rangle \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
 $\rangle \wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$
using $h2 \text{ ImpliesNot.prem } \text{by simp}$
from this obtain $\Gamma_k \Psi_k \Phi_k k$ **where**
 $fp: \langle (((K_1 \uparrow n) \# (K_2 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \# \Phi))$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
and $rc: \langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$ **by blast**
have $\langle (\Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi)$
 $\hookrightarrow (((K_1 \uparrow n) \# (K_2 \neg\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ implies not } K_2) \#$
 $\Phi)) \rangle$
by ($\text{simp add: elims-part implies-not-e2}$)
hence $\langle (\Gamma, n \vdash ((K_1 \text{ implies not } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k$
 $\triangleright \Phi_k) \rangle$
using $fp \text{ relpowp-Suc-I2}$ **by auto**
with rc show $?thesis$ **by blast**
qed
ultimately show $?case$ **using** $\text{ImpliesNot.prem}(2)$ **by blast**
next
case ($\text{TimeDelayedBy } K_1 \delta\tau K_2 K_3$)
have branches: $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \rrbracket$

$\triangleright \Phi \llbracket_{config}$
 $= \llbracket ((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rrbracket_{config}$
 $\cup \llbracket ((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rrbracket_{config}$
using *HeronConf-interp-stepwise-timedelayed-cases* **by** *simp*
moreover have *br1*: $\langle \varrho \in \llbracket ((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rrbracket_{config} \rangle$
 $\implies \exists \Gamma_k \Psi_k \Phi_k k.$
 $((\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi) \hookrightarrow^k$
 $(\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
 $\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config}$
proof –
assume *h1*: $\langle \varrho \in \llbracket ((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rrbracket_{config} \rangle$
then have $\langle \exists \Gamma_k \Psi_k \Phi_k k. \rangle$
 $((((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi))$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
 $\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config}$
using *h1 TimeDelayedBy.premis* **by** *simp*
from this obtain $\Gamma_k \Psi_k \Phi_k k$
where *fp*: $\langle (((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi))$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
and *rc*: $\langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config} \rangle$ **by** *blast*
have $\langle (\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi) \rangle$
 $\hookrightarrow (((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi)) \rangle$
by (*simp add: elims-part timedelayed-e1*)
hence $\langle (\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi) \rangle$
 $\hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
using *fp relpowp-Suc-I2* **by** *auto*
with rc show *?thesis* **by** *blast*
qed
moreover have *br2*:
 $\langle \varrho \in \llbracket ((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rrbracket_{config} \rangle$
 $\implies \exists \Gamma_k \Psi_k \Phi_k k.$
 $((\Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi)$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
 $\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config}$
proof –
assume *h2*: $\langle \varrho \in \llbracket ((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rrbracket_{config} \rangle$
then have $\langle \exists \Gamma_k \Psi_k \Phi_k k. (((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi))$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$

$\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}}$
using *h2 TimeDelayedBy.premis by simp*
from this obtain $\Gamma_k \Psi_k \Phi_k k$
where $\text{fp} : \langle ((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rangle$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
and $\text{rc} : \langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$ **by** *blast*
have $\langle \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow \langle ((K_1 \uparrow n) \# (K_2 @ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Phi) \rangle$
by (*simp add: elims-part timedelayed-e2*)
with *fp relpowp-Suc-I2* **have**
 $\langle \Gamma, n \vdash ((K_1 \text{ time-delayed by } \delta\tau \text{ on } K_2 \text{ implies } K_3) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
by *auto*
with *rc* **show** *?thesis* **by** *blast*
qed
ultimately show *?case* **using** *TimeDelayedBy.premis(2)* **by** *blast*
next
case (*WeaklyPrecedes* $K_1 K_2$)
have $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi \rrbracket_{\text{config}} =$
 $\llbracket ((\lceil \# \leq K_2 n, \# \leq K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ weakly}$
precedes $K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
using *HeronConf-interp-stepwise-weakly-precedes-cases* **by** *simp*
moreover have $\langle \varrho \in \llbracket ((\lceil \# \leq K_2 n, \# \leq K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \rrbracket_{\text{config}}$
 $\implies (\exists \Gamma_k \Psi_k \Phi_k k. (\Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi)$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
 $\wedge (\varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}}) \rangle$
proof –
assume $\langle \varrho \in \llbracket ((\lceil \# \leq K_2 n, \# \leq K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
hence $\exists \Gamma_k \Psi_k \Phi_k k. (\langle ((\lceil \# \leq K_2 n, \# \leq K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma),$
 n
 $\vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \rangle$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle \wedge (\varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k$
 $\rrbracket_{\text{config}}) \rangle$
using *WeaklyPrecedes.premis* **by** *simp*
from this obtain $\Gamma_k \Psi_k \Phi_k k$
where $\text{fp} : \langle ((\lceil \# \leq K_2 n, \# \leq K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \rangle$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
and $\text{rc} : \langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$ **by** *blast*
have $\langle \Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow \langle ((\lceil \# \leq K_2 n, \# \leq K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ weakly precedes } K_2) \# \Phi) \rangle$ **by** (*simp add: elims-part*
weakly-precedes-e)
with *fp relpowp-Suc-I2* **have** $\langle \Gamma, n \vdash ((K_1 \text{ weakly precedes } K_2) \# \Psi) \triangleright \Phi \rangle$
 $\hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$

by *auto*
 with *rc* show ?thesis by *blast*
 qed
 ultimately show ?case using *WeaklyPrecedes.prem(2)* by *blast*
 next
 case (*StrictlyPrecedes* $K_1 K_2$)
 have $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi \rrbracket_{\text{config}} =$
 $\llbracket ((\lceil \#^{\leq} K_2 n, \#^< K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ strictly}$
precedes $K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
 using *HeronConf-interp-stepwise-strictly-precedes-cases* by *simp*
 moreover have $\langle \varrho \in \llbracket ((\lceil \#^{\leq} K_2 n, \#^< K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi) \rrbracket_{\text{config}}$
 $\implies (\exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi)$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
 $\wedge (\varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}})) \rangle$
 proof –
 assume $\langle \varrho \in \llbracket ((\lceil \#^{\leq} K_2 n, \#^< K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi) \rrbracket_{\text{config}}$
 hence $\langle \exists \Gamma_k \Psi_k \Phi_k k. (((\lceil \#^{\leq} K_2 n, \#^< K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma),$
 n
 $\vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi))$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \wedge (\varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k$
 $\rrbracket_{\text{config}}) \rangle$
 using *StrictlyPrecedes.prem(2)* by *simp*
 from this obtain $\Gamma_k \Psi_k \Phi_k k$
 where $fp: \langle (((\lceil \#^{\leq} K_2 n, \#^< K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi))$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
 and $rc: \langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$ by *blast*
 have $\langle (\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi)$
 $\hookrightarrow (((\lceil \#^{\leq} K_2 n, \#^< K_1 n \rceil \in (\lambda(x, y). x \leq y)) \# \Gamma), n$
 $\vdash \Psi \triangleright ((K_1 \text{ strictly precedes } K_2) \# \Phi)) \rangle$ by (*simp add: elims-part*
strictly-precedes-e)
 with *fp relpow-Suc-I2* have $\langle (\Gamma, n \vdash ((K_1 \text{ strictly precedes } K_2) \# \Psi) \triangleright \Phi)$
 $\hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
 by *auto*
 with *rc* show ?thesis by *blast*
 qed
 ultimately show ?case using *StrictlyPrecedes.prem(2)* by *blast*
 next
 case (*Kills* $K_1 K_2$)
 have branches: $\langle \llbracket \Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi \rrbracket_{\text{config}}$
 $= \llbracket ((K_1 \dashv\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rrbracket_{\text{config}}$
 $\cup \llbracket ((K_1 \uparrow n) \# (K_2 \dashv\uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
 using *HeronConf-interp-stepwise-kills-cases* by *simp*
 moreover have $br1: \langle \varrho \in \llbracket ((K_1 \dashv\uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)$
 $\rrbracket_{\text{config}}$
 $\implies \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi)$
 $\hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k))$

$\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}}$
proof –
assume $h1: \langle \varrho \in \llbracket ((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
then have $\langle \exists \Gamma_k \Psi_k \Phi_k k.$
 $((((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \hookrightarrow^k (\Gamma_k, \text{Suc } n$
 $\vdash \Psi_k \triangleright \Phi_k)) \rangle$
 $\wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}}$
using $h1 \text{ Kills.prem } \text{by simp}$
from this obtain $\Gamma_k \Psi_k \Phi_k k$ **where**
 $fp: \langle (((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash$
 $\Psi_k \triangleright \Phi_k) \rangle$
and $rc: \langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$ **by blast**
have $pc: \langle (\Gamma, n \vdash (K_1 \text{ kills } K_2) \# \Psi \triangleright \Phi) \hookrightarrow (((K_1 \neg \uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \rangle$
by $(\text{simp add: elims-part kills-e1})$
hence $\langle (\Gamma, n \vdash (K_1 \text{ kills } K_2) \# \Psi \triangleright \Phi) \hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
using $fp \text{ relpowp-Suc-I2}$ **by auto**
with rc show $?thesis$ **by blast**
qed
moreover have $br2: \langle \varrho \in \llbracket ((K_1 \uparrow n) \# (K_2 \neg \uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1$
 $\text{kills } K_2) \# \Phi) \rrbracket_{\text{config}} \implies \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow^k$
 $(\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$
proof –
assume $h2: \langle \varrho \in \llbracket ((K_1 \uparrow n) \# (K_2 \neg \uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rrbracket_{\text{config}} \rangle$
then have $\langle \exists \Gamma_k \Psi_k \Phi_k k. ($
 $((K_1 \uparrow n) \# (K_2 \neg \uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi) \rangle \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
 $\rangle \wedge \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}}$
using $h2 \text{ Kills.prem } \text{by simp}$
from this obtain $\Gamma_k \Psi_k \Phi_k k$ **where**
 $fp: \langle (((K_1 \uparrow n) \# (K_2 \neg \uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \hookrightarrow^k (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
and $rc: \langle \varrho \in \llbracket \Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{\text{config}} \rangle$ **by blast**
have $\langle (\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow (((K_1 \uparrow n) \# (K_2 \neg \uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \text{ kills } K_2) \# \Phi)) \rangle$
by $(\text{simp add: elims-part kills-e2})$
hence $\langle (\Gamma, n \vdash ((K_1 \text{ kills } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow^{\text{Suc } k} (\Gamma_k, \text{Suc } n \vdash \Psi_k \triangleright \Phi_k) \rangle$
using $fp \text{ relpowp-Suc-I2}$ **by auto**
with rc show $?thesis$ **by blast**
qed
ultimately show $?case$ **using** $\text{Kills.prem}(2)$ **by blast**
qed
qed

lemma *instant-index-increase-generalized:*

assumes $\langle n < n_k \rangle$

assumes $\langle \varrho \in \llbracket \Gamma, n \vdash \Psi \triangleright \Phi \rrbracket_{config} \rangle$
shows $\langle \exists \Gamma_k \Psi_k \Phi_k k. ((\Gamma, n \vdash \Psi \triangleright \Phi) \hookrightarrow^k (\Gamma_k, n_k \vdash \Psi_k \triangleright \Phi_k))$
 $\quad \wedge \varrho \in \llbracket \Gamma_k, n_k \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config} \rangle$
proof –
obtain δk **where** $diff: \langle n_k = \delta k + Suc\ n \rangle$
using *add.commute assms(1) less-iff-Suc-add* **by** *auto*
show *?thesis*
proof (*subst diff, subst diff, insert assms(2), induct δk*)
case 0
then show *?case*
using *instant-index-increase assms(2)* **by** *simp*
next
case (*Suc δk*)
have $f0: \langle \varrho \in \llbracket \Gamma, n \vdash \Psi \triangleright \Phi \rrbracket_{config} \implies \exists \Gamma_k \Psi_k \Phi_k k.$
 $\quad ((\Gamma, n \vdash \Psi \triangleright \Phi) \hookrightarrow^k (\Gamma_k, \delta k + Suc\ n \vdash \Psi_k \triangleright \Phi_k))$
 $\quad \wedge \varrho \in \llbracket \Gamma_k, \delta k + Suc\ n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config} \rangle$
using *Suc.hyyps* **by** *blast*
obtain $\Gamma_k \Psi_k \Phi_k k$
where $cont: \langle ((\Gamma, n \vdash \Psi \triangleright \Phi) \hookrightarrow^k (\Gamma_k, \delta k + Suc\ n \vdash \Psi_k \triangleright \Phi_k)) \wedge \varrho \in \llbracket$
 $\Gamma_k, \delta k + Suc\ n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config} \rangle$
using $f0$ *assms(1) Suc.prem*s **by** *blast*
then have $fcontinue: \langle \exists \Gamma_k' \Psi_k' \Phi_k' k'. ((\Gamma_k, \delta k + Suc\ n \vdash \Psi_k \triangleright \Phi_k) \hookrightarrow^{k'}$
 $(\Gamma_k', Suc\ (\delta k + Suc\ n) \vdash \Psi_k' \triangleright \Phi_k'))$
 $\quad \wedge \varrho \in \llbracket \Gamma_k', Suc\ (\delta k + Suc\ n) \vdash \Psi_k' \triangleright \Phi_k' \rrbracket_{config} \rangle$
using $f0$ *cont instant-index-increase* **by** *blast*
obtain $\Gamma_k' \Psi_k' \Phi_k' k'$ **where** $cont2: \langle ((\Gamma_k, \delta k + Suc\ n \vdash \Psi_k \triangleright \Phi_k) \hookrightarrow^{k'} (\Gamma_k',$
 $Suc\ (\delta k + Suc\ n) \vdash \Psi_k' \triangleright \Phi_k'))$
 $\quad \wedge \varrho \in \llbracket \Gamma_k', Suc\ (\delta k + Suc\ n) \vdash \Psi_k' \triangleright \Phi_k' \rrbracket_{config} \rangle$
using *Suc.prem*s **using** $fcontinue$ *cont* **by** *blast*
have $trans: \langle (\Gamma, n \vdash \Psi \triangleright \Phi) \hookrightarrow^{k+k'} (\Gamma_k', Suc\ (\delta k + Suc\ n) \vdash \Psi_k' \triangleright \Phi_k') \rangle$
using *operational-semantics-trans-generalized cont cont2*
by *blast*
moreover have *suc-assoc*: $\langle Suc\ \delta k + Suc\ n = Suc\ (\delta k + Suc\ n) \rangle$
by *arith*
ultimately show *?case*
proof (*subst suc-assoc*)
show $\langle \exists \Gamma_k \Psi_k \Phi_k k.$
 $\quad ((\Gamma, n \vdash \Psi \triangleright \Phi) \hookrightarrow^k (\Gamma_k, Suc\ (\delta k + Suc\ n) \vdash \Psi_k \triangleright \Phi_k))$
 $\quad \wedge \varrho \in \llbracket \Gamma_k, Suc\ \delta k + Suc\ n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config} \rangle$
using $cont2$ *local.trans* **by** *auto*
qed
qed

Any run from initial specification Ψ has a corresponding configuration indexed at n -th instant starting from initial configuration.

theorem *progress*:

assumes $\langle \varrho \in \llbracket \Psi \rrbracket_{TESL} \rangle$
shows $\langle \exists k \Gamma_k \Psi_k \Phi_k. ((\Box, 0 \vdash \Psi \triangleright \Box) \hookrightarrow^k (\Gamma_k, n \vdash \Psi_k \triangleright \Phi_k)) \wedge \varrho \in \llbracket \Gamma_k, n \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config} \rangle$
proof –
have $1: \langle \exists \Gamma_k \Psi_k \Phi_k k. ((\Box, 0 \vdash \Psi \triangleright \Box) \hookrightarrow^k (\Gamma_k, 0 \vdash \Psi_k \triangleright \Phi_k)) \wedge \varrho \in \llbracket \Gamma_k, 0 \vdash \Psi_k \triangleright \Phi_k \rrbracket_{config} \rangle$
using *assms relpowp-0-I solve-start* **by** *fastforce*
show *?thesis*
proof (*cases* $\langle n = 0 \rangle$)
case *True*
thus *?thesis* **using** *assms relpowp-0-I solve-start* **by** *fastforce*
next
case *False* **hence** *pos: $\langle n > 0 \rangle$* **by** *simp*
from *assms solve-start* **have** $\langle \varrho \in \llbracket \Box, 0 \vdash \Psi \triangleright \Box \rrbracket_{config} \rangle$ **by** *blast*
from *instant-index-increase-generalized[OF pos this]* **show** *?thesis* **by** *blast*
qed
qed

6.5 Local termination

primrec *measure-interpretation* :: $\langle ' \tau :: \text{linordered-field TESL-formula} \Rightarrow \text{nat} \rangle (\mu)$
where

$\langle \mu \Box = (0::\text{nat}) \rangle$
 $\mid \langle \mu (\varphi \# \Phi) = (\text{case } \varphi \text{ of}$
 $\quad - \text{ sporadic } - \text{ on } - \Rightarrow 1 + \mu \Phi$
 $\quad \mid - \Rightarrow 2 + \mu \Phi) \rangle$

fun *measure-interpretation-config* :: $\langle ' \tau :: \text{linordered-field config} \Rightarrow \text{nat} \rangle (\mu_{config})$
where

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

lemma *elimination-rules-strictly-decreasing*:

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

shows $\langle \mu \Psi_1 > \mu \Psi_2 \rangle$

by (*insert assms, erule operational-semantics-elim.cases, auto*)

lemma *elimination-rules-strictly-decreasing-meas*:

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

shows $\langle (\Psi_2, \Psi_1) \in \text{measure } \mu \rangle$

by (*insert assms, erule operational-semantics-elim.cases, auto*)

lemma *elimination-rules-strictly-decreasing-meas'*:

assumes $\langle \mathcal{S}_1 \hookrightarrow_e \mathcal{S}_2 \rangle$

shows $\langle (\mathcal{S}_2, \mathcal{S}_1) \in \text{measure } \mu_{config} \rangle$

proof –

from *assms* **obtain** $\Gamma_1 \ n_1 \ \Psi_1 \ \Phi_1$ **where** $p1: \langle \mathcal{S}_1 = (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \rangle$

using *measure-interpretation-config.cases* **by** *blast*

from *assms* **obtain** $\Gamma_2 \ n_2 \ \Psi_2 \ \Phi_2$ **where** $p2: \langle \mathcal{S}_2 = (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) \rangle$

using *measure-interpretation-config.cases* **by** *blast*

from *elimination-rules-strictly-decreasing-meas* *assms* *p1 p2*
have $\langle (\Psi_2, \Psi_1) \in \text{measure } \mu \rangle$ **by** *blast*
hence $\langle \mu \Psi_2 < \mu \Psi_1 \rangle$ **by** *simp*
hence $\langle \mu_{\text{config}} (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) < \mu_{\text{config}} (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \rangle$ **by** *simp*
with *p1 p2* **show** *?thesis* **by** *simp*
qed

The relation made up of elimination rules is well-founded.

theorem *instant-computation-termination:*

shows $\langle \text{wfP } (\lambda(\mathcal{S}_1 :: 'a :: \text{linordered-field config}) \mathcal{S}_2. (\mathcal{S}_1 \hookrightarrow_e^{\leftarrow} \mathcal{S}_2)) \rangle$
proof (*simp add: wfP-def*)
show $\langle \text{wf } \{((\mathcal{S}_1 :: 'a :: \text{linordered-field config}), \mathcal{S}_2). \mathcal{S}_1 \hookrightarrow_e^{\leftarrow} \mathcal{S}_2\} \rangle$
proof (*rule wf-subset*)
have $\langle \text{measure } \mu_{\text{config}} = \{ (\mathcal{S}_2, (\mathcal{S}_1 :: 'a :: \text{linordered-field config})). \mu_{\text{config}} \mathcal{S}_1 \} \rangle$
by (*simp add: inv-image-def less-eq measure-def*)
thus $\langle \{((\mathcal{S}_1 :: 'a :: \text{linordered-field config}), \mathcal{S}_2). \mathcal{S}_1 \hookrightarrow_e^{\leftarrow} \mathcal{S}_2\} \subseteq (\text{measure } \mu_{\text{config}}) \rangle$
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 7

Properties of TESL

7.1 Stuttering Invariance

theory *StutteringDefs*

imports *Denotational*

begin

7.1.1 Definition of stuttering

A dilating function inserts empty instants in a run. It is strictly increasing, the image of a *nat* is greater than it, no instant is inserted before the first one and if *n* is not in the image of the function, no clock ticks at instant *n*.

definition *dilating-fun*

where

$$\langle \text{dilating-fun } (f :: \text{nat} \Rightarrow \text{nat}) \ (r :: 'a :: \text{linordered-field} \ \text{run}) \rangle$$
$$\equiv \text{strict-mono } f \wedge (f \ 0 = 0) \wedge (\forall n. f \ n \geq n$$
$$\wedge ((\nexists n_0. f \ n_0 = n) \longrightarrow (\forall c. \neg(\text{hamlet } ((\text{Rep-run } r) \ n \ c))))$$
$$\wedge ((\nexists n_0. f \ n_0 = (\text{Suc } n)) \longrightarrow (\forall c. \text{time } ((\text{Rep-run } r) \ (\text{Suc } n) \ c) = \text{time}$$
$$((\text{Rep-run } r) \ n \ c)))$$
$$\rangle$$

Dilating a run. A run *r* is a dilation of a run *sub* by function *f* if:

- *f* is a dilating function on the hamlet of *r*
- time is preserved in stuttering instants
- the time in *r* is the time in *sub* dilated by *f*
- the hamlet in *r* is the hamlet in *sub* dilated by *f*

definition *dilating*

where $\langle \text{dilating } f \text{ sub } r \equiv \text{dilating-fun } f \text{ } r$
 $\wedge (\forall n \text{ } c. \text{time } ((\text{Rep-run sub}) \text{ } n \text{ } c) = \text{time } ((\text{Rep-run } r) \text{ } (f$
 $n) \text{ } c))$
 $\wedge (\forall n \text{ } c. \text{hamlet } ((\text{Rep-run sub}) \text{ } n \text{ } c) = \text{hamlet } ((\text{Rep-run}$
 $r) \text{ } (f \text{ } n) \text{ } c)) \rangle$

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

definition *is-subrun* :: $\langle 'a::\text{linordered-field run} \Rightarrow 'a \text{ run} \Rightarrow \text{bool} \rangle$ (**infixl** $\ll 60$)
where

$\langle \text{sub} \ll r \equiv (\exists f. \text{dilating } f \text{ sub } r) \rangle$

A *tick-count* $r \text{ } c \text{ } n$ is a number of ticks of clock c in run r upto instant n .

definition *tick-count* :: $\langle 'a::\text{linordered-field run} \Rightarrow \text{clock} \Rightarrow \text{nat} \Rightarrow \text{nat} \rangle$
where

$\langle \text{tick-count } r \text{ } c \text{ } n = \text{card } \{i. i \leq n \wedge \text{hamlet } ((\text{Rep-run } r) \text{ } i \text{ } c)\} \rangle$

A *tick-count-strict* $r \text{ } c \text{ } n$ is a number of ticks of clock c in run r upto but excluding instant n .

definition *tick-count-strict* :: $\langle 'a::\text{linordered-field run} \Rightarrow \text{clock} \Rightarrow \text{nat} \Rightarrow \text{nat} \rangle$
where

$\langle \text{tick-count-strict } r \text{ } c \text{ } n = \text{card } \{i. i < n \wedge \text{hamlet } ((\text{Rep-run } r) \text{ } i \text{ } c)\} \rangle$

definition *contracting-fun*

where $\langle \text{contracting-fun } g \equiv \text{mono } g \wedge g \text{ } 0 = 0 \wedge (\forall n. g \text{ } n \leq n) \rangle$

definition *contracting*

where

$\langle \text{contracting } g \text{ } r \text{ sub } f \equiv \text{contracting-fun } g$
 $\wedge (\forall n \text{ } c \text{ } k. f \text{ } (g \text{ } n) \leq k \wedge k \leq n$
 $\longrightarrow \text{time } ((\text{Rep-run } r) \text{ } k \text{ } c) = \text{time } ((\text{Rep-run sub}) \text{ } (g \text{ } n) \text{ } c))$
 $\wedge (\forall n \text{ } c \text{ } k. f \text{ } (g \text{ } n) < k \wedge k \leq n$
 $\longrightarrow \neg \text{hamlet } ((\text{Rep-run } r) \text{ } k \text{ } c)) \rangle$

definition $\langle \text{dil-inverse } f::(\text{nat} \Rightarrow \text{nat}) \equiv (\lambda n. \text{Max } \{i. f \text{ } i \leq n\}) \rangle$

end

7.1.2 Stuttering Lemmas

theory *StutteringLemmas*

imports *StutteringDefs*

begin

lemma *bounded-suc-ind*:

assumes $\langle \bigwedge k. k < m \Longrightarrow P \text{ } (\text{Suc } (z + k)) = P \text{ } (z + k) \rangle$

shows $\langle k < m \Longrightarrow P \text{ } (\text{Suc } (z + k)) = P \text{ } z \rangle$

proof (*induction k*)

```

case 0
  with assms(1)[of 0] show ?case by simp
next
case (Suc k')
  with assms[of (Suc k')] show ?case by force
qed

```

7.1.3 Lemmas used to prove the invariance by stuttering

A dilating function is injective.

```

lemma dilating-fun-injects:
  assumes (dilating-fun f r)
  shows (inj-on f A)
using assms dilating-fun-def strict-mono-imp-inj-on 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 (dilating-fun f r)
  and (hamlet ((Rep-run r) n c))
  shows (∃ n0. f n0 = n)
using dilating-fun-def assms by blast

```

The image of the ticks in a interval by a dilating function is the interval bounded by the image of the bound 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 (dilating-fun f r)
  shows {k. f m < k ∧ k < f n ∧ hamlet ((Rep-run r) k c)}
        = image f {k. m < k ∧ k < n ∧ hamlet ((Rep-run r) (f k) c)}
  (is (⟨?IMG = image f ?SET⟩))
proof
  { fix k assume h:⟨k ∈ ?IMG⟩
    from h obtain k0 where k0prop:⟨f k0 = k ∧ hamlet ((Rep-run r) (f k0) c)⟩
    using ticks-image[OF assms] by blast
    with h have ⟨k ∈ image f ?SET⟩ using assms dilating-fun-def strict-mono-less
  by blast
  } thus ⟨?IMG ⊆ image f ?SET⟩ ..
next
  { fix k assume h:⟨k ∈ image f ?SET⟩
    from h obtain k0 where k0prop:⟨k = f k0 ∧ k0 ∈ ?SET⟩ by blast
    hence ⟨k ∈ ?IMG⟩ using assms by (simp add: dilating-fun-def strict-mono-less)
  } thus ⟨image f ?SET ⊆ ?IMG⟩ ..
qed

```

```

lemma dilating-fun-image-left:
  assumes (dilating-fun f r)
  shows {k. f m ≤ k ∧ k < f n ∧ hamlet ((Rep-run r) k c)}

```

$= \text{image } f \{k. m \leq k \wedge k < n \wedge \text{hamlet } ((\text{Rep-run } r) (f k) c)\}$
 (is $\langle ?IMG = \text{image } f ?SET \rangle$)

proof

{ **fix** k **assume** $h:\langle k \in ?IMG \rangle$
 from h **obtain** k_0 **where** $k_0\text{prop}:\langle f k_0 = k \wedge \text{hamlet } ((\text{Rep-run } r) (f k_0) c) \rangle$
 using $\text{ticks-image}[OF \text{ assms}]$ **by** blast
 with h **have** $\langle k \in \text{image } f ?SET \rangle$
 using $\text{assms dilating-fun-def strict-mono-less strict-mono-less-eq}$ **by** fastforce
 } **thus** $\langle ?IMG \subseteq \text{image } f ?SET \rangle$..

next

{ **fix** k **assume** $h:\langle k \in \text{image } f ?SET \rangle$
 from h **obtain** k_0 **where** $k_0\text{prop}:\langle k = f k_0 \wedge k_0 \in ?SET \rangle$ **by** blast
 hence $\langle k \in ?IMG \rangle$
 using $\text{assms dilating-fun-def strict-mono-less strict-mono-less-eq}$ **by** fastforce
 } **thus** $\langle \text{image } f ?SET \subseteq ?IMG \rangle$..

qed

lemma *dilating-fun-image-right*:

assumes $\langle \text{dilating-fun } f r \rangle$
shows $\langle \{k. f m < k \wedge k \leq f n \wedge \text{hamlet } ((\text{Rep-run } r) k c)\}$
 $= \text{image } f \{k. m < k \wedge k \leq n \wedge \text{hamlet } ((\text{Rep-run } r) (f k) c)\}$
 (is $\langle ?IMG = \text{image } f ?SET \rangle$)

proof

{ **fix** k **assume** $h:\langle k \in ?IMG \rangle$
 from h **obtain** k_0 **where** $k_0\text{prop}:\langle f k_0 = k \wedge \text{hamlet } ((\text{Rep-run } r) (f k_0) c) \rangle$
 using $\text{ticks-image}[OF \text{ assms}]$ **by** blast
 with h **have** $\langle k \in \text{image } f ?SET \rangle$
 using $\text{assms dilating-fun-def strict-mono-less strict-mono-less-eq}$ **by** fastforce
 } **thus** $\langle ?IMG \subseteq \text{image } f ?SET \rangle$..

next

{ **fix** k **assume** $h:\langle k \in \text{image } f ?SET \rangle$
 from h **obtain** k_0 **where** $k_0\text{prop}:\langle k = f k_0 \wedge k_0 \in ?SET \rangle$ **by** blast
 hence $\langle k \in ?IMG \rangle$
 using $\text{assms dilating-fun-def strict-mono-less strict-mono-less-eq}$ **by** fastforce
 } **thus** $\langle \text{image } f ?SET \subseteq ?IMG \rangle$..

qed

lemma *dilating-fun-image*:

assumes $\langle \text{dilating-fun } f r \rangle$
shows $\langle \{k. f m \leq k \wedge k \leq f n \wedge \text{hamlet } ((\text{Rep-run } r) k c)\}$
 $= \text{image } f \{k. m \leq k \wedge k \leq n \wedge \text{hamlet } ((\text{Rep-run } r) (f k) c)\}$
 (is $\langle ?IMG = \text{image } f ?SET \rangle$)

proof

{ **fix** k **assume** $h:\langle k \in ?IMG \rangle$
 from h **obtain** k_0 **where** $k_0\text{prop}:\langle f k_0 = k \wedge \text{hamlet } ((\text{Rep-run } r) (f k_0) c) \rangle$
 using $\text{ticks-image}[OF \text{ assms}]$ **by** blast
 with h **have** $\langle k \in \text{image } f ?SET \rangle$
 using $\text{assms dilating-fun-def strict-mono-less-eq}$ **by** blast
 } **thus** $\langle ?IMG \subseteq \text{image } f ?SET \rangle$..

next
 { **fix** k **assume** $h: \langle k \in \text{image } f \text{ ?SET} \rangle$
 from h **obtain** k_0 **where** $k_0 \text{prop}: \langle k = f k_0 \wedge k_0 \in \text{?SET} \rangle$ **by** *blast*
 hence $\langle k \in \text{?IMG} \rangle$ **using** *assms* **by** (*simp add: dilating-fun-def strict-mono-less-eq*)
 } **thus** $\langle \text{image } f \text{ ?SET} \subseteq \text{?IMG} \rangle$..
qed

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

lemma *ticks-as-often-strict:*

assumes $\langle \text{dilating-fun } f \text{ } r \rangle$
shows $\langle \text{card } \{p. n < p \wedge p < m \wedge \text{hamlet } ((\text{Rep-run } r) (f p) c)\} \rangle$
 $= \text{card } \{p. f n < p \wedge p < f m \wedge \text{hamlet } ((\text{Rep-run } r) p c)\} \rangle$
 (is $\langle \text{card } \text{?SET} = \text{card } \text{?IMG} \rangle$)
proof –
 from *dilating-fun-injects*[*OF assms*] **have** $\langle \text{inj-on } f \text{ ?SET} \rangle$.
 moreover **have** $\langle \text{finite } \text{?SET} \rangle$ **by** *simp*
 from *inj-on-iff-eq-card*[*OF this*] **calculation** **have** $\langle \text{card } (\text{image } f \text{ ?SET}) = \text{card } \text{?SET} \rangle$ **by** *blast*
 moreover **from** *dilating-fun-image-strict*[*OF assms*] **have** $\langle \text{?IMG} = \text{image } f \text{ ?SET} \rangle$.
 ultimately show *?thesis* **by** *auto*
qed

lemma *ticks-as-often-left:*

assumes $\langle \text{dilating-fun } f \text{ } r \rangle$
shows $\langle \text{card } \{p. n \leq p \wedge p < m \wedge \text{hamlet } ((\text{Rep-run } r) (f p) c)\} \rangle$
 $= \text{card } \{p. f n \leq p \wedge p < f m \wedge \text{hamlet } ((\text{Rep-run } r) p c)\} \rangle$
 (is $\langle \text{card } \text{?SET} = \text{card } \text{?IMG} \rangle$)
proof –
 from *dilating-fun-injects*[*OF assms*] **have** $\langle \text{inj-on } f \text{ ?SET} \rangle$.
 moreover **have** $\langle \text{finite } \text{?SET} \rangle$ **by** *simp*
 from *inj-on-iff-eq-card*[*OF this*] **calculation** **have** $\langle \text{card } (\text{image } f \text{ ?SET}) = \text{card } \text{?SET} \rangle$ **by** *blast*
 moreover **from** *dilating-fun-image-left*[*OF assms*] **have** $\langle \text{?IMG} = \text{image } f \text{ ?SET} \rangle$
 .
 ultimately show *?thesis* **by** *auto*
qed

lemma *ticks-as-often-right:*

assumes $\langle \text{dilating-fun } f \text{ } r \rangle$
shows $\langle \text{card } \{p. n < p \wedge p \leq m \wedge \text{hamlet } ((\text{Rep-run } r) (f p) c)\} \rangle$
 $= \text{card } \{p. f n < p \wedge p \leq f m \wedge \text{hamlet } ((\text{Rep-run } r) p c)\} \rangle$
 (is $\langle \text{card } \text{?SET} = \text{card } \text{?IMG} \rangle$)
proof –
 from *dilating-fun-injects*[*OF assms*] **have** $\langle \text{inj-on } f \text{ ?SET} \rangle$.
 moreover **have** $\langle \text{finite } \text{?SET} \rangle$ **by** *simp*
 from *inj-on-iff-eq-card*[*OF this*] **calculation** **have** $\langle \text{card } (\text{image } f \text{ ?SET}) = \text{card } \text{?SET} \rangle$ **by** *blast*

moreover from *dilating-fun-image-right*[*OF assms*] **have** $\langle ?IMG = image\ f\ ?SET \rangle$.

ultimately show *?thesis* **by** *auto*
qed

lemma *ticks-as-often*:

assumes $\langle dilating\text{-}fun\ f\ r \rangle$

shows $\langle card\ \{p. n \leq p \wedge p \leq m \wedge hamlet\ ((Rep\text{-}run\ r)\ (f\ p)\ c)\} \\ = card\ \{p. f\ n \leq p \wedge p \leq f\ m \wedge hamlet\ ((Rep\text{-}run\ r)\ p\ c)\} \rangle$
 $(is\ \langle card\ ?SET = card\ ?IMG \rangle)$

proof –

from *dilating-fun-injects*[*OF assms*] **have** $\langle inj\text{-}on\ f\ ?SET \rangle$.

moreover have $\langle finite\ ?SET \rangle$ **by** *simp*

from *inj-on-iff-eq-card*[*OF this*] **calculation have** $\langle card\ (image\ f\ ?SET) = card\ ?SET \rangle$ **by** *blast*

moreover from *dilating-fun-image*[*OF assms*] **have** $\langle ?IMG = image\ f\ ?SET \rangle$.

ultimately show *?thesis* **by** *auto*
qed

lemma *dilating-injects*:

assumes $\langle dilating\ f\ sub\ r \rangle$

shows $\langle inj\text{-}on\ f\ A \rangle$

using *assms* **by** (*simp add: dilating-def dilating-fun-def strict-mono-imp-inj-on*)

If there is a tick at instant *n* in a dilated run, *n* is necessarily the image of some instant in the subrun.

lemma *ticks-image-sub*:

assumes $\langle dilating\ f\ sub\ r \rangle$

and $\langle hamlet\ ((Rep\text{-}run\ r)\ n\ c) \rangle$

shows $\langle \exists n_0. f\ n_0 = n \rangle$

proof –

from *assms(1)* **have** $\langle dilating\text{-}fun\ f\ r \rangle$ **by** (*simp add: dilating-def*)

from *ticks-image*[*OF this assms(2)*] **show** *?thesis* .

qed

lemma *ticks-image-sub'*:

assumes $\langle dilating\ f\ sub\ r \rangle$

and $\langle \exists c. hamlet\ ((Rep\text{-}run\ r)\ n\ c) \rangle$

shows $\langle \exists n_0. f\ n_0 = n \rangle$

proof –

from *assms(1)* **have** $\langle dilating\text{-}fun\ f\ r \rangle$ **by** (*simp add: dilating-def*)

with *dilating-fun-def assms(2)* **show** *?thesis* **by** *blast*

qed

Time is preserved by dilation when ticks occur.

lemma *ticks-tag-image*:

assumes $\langle dilating\ f\ sub\ r \rangle$

and $\langle \exists c. hamlet\ ((Rep\text{-}run\ r)\ k\ c) \rangle$

and $\langle time\ ((Rep\text{-}run\ r)\ k\ c) = \tau \rangle$

shows $\langle \exists k_0. f k_0 = k \wedge \text{time } ((\text{Rep-run sub}) k_0 c) = \tau \rangle$
proof –
from *ticks-image-sub*[*OF assms*(1,2)] **have** $\langle \exists k_0. f k_0 = k \rangle$.
from this obtain k_0 **where** $\langle f k_0 = k \rangle$ **by** *blast*
moreover with *assms*(1,3) **have** $\langle \text{time } ((\text{Rep-run sub}) k_0 c) = \tau \rangle$ **by** (*simp*
add: dilating-def)
ultimately show *?thesis* **by** *blast*
qed

TESL operators are preserved by dilation.

lemma *ticks-sub*:
assumes $\langle \text{dilating } f \text{ sub } r \rangle$
shows $\langle \text{hamlet } ((\text{Rep-run sub}) n a) = \text{hamlet } ((\text{Rep-run } r) (f n) a) \rangle$
using *assms* **by** (*simp add: dilating-def*)

lemma *no-tick-sub*:
assumes $\langle \text{dilating } f \text{ sub } r \rangle$
shows $\langle (\nexists n_0. f n_0 = n) \longrightarrow \neg \text{hamlet } ((\text{Rep-run } r) n a) \rangle$
using *assms* *dilating-def* *dilating-fun-def* **by** *blast*

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

definition *opt-lift*:: $\langle 'a \Rightarrow 'a \rangle \Rightarrow \langle 'a \text{ option} \Rightarrow 'a \text{ option} \rangle$
where
 $\langle \text{opt-lift } f \equiv \lambda x. \text{ case } x \text{ of } \text{None} \Rightarrow \text{None} \mid \text{Some } y \Rightarrow \text{Some } (f y) \rangle$

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 $\langle \text{dilating } f \text{ sub } r \rangle$
shows $\langle \{k. \text{hamlet } ((\text{Rep-run } r) k c)\} = \text{image } f \{k. \text{hamlet } ((\text{Rep-run sub}) k c)\} \rangle$
 $(\text{is } \langle ?R = \text{image } f ?S \rangle)$

proof
{ fix k **assume** $h:\langle k \in ?R \rangle$
with *no-tick-sub*[*OF assms*] **have** $\langle \exists k_0. f k_0 = k \rangle$ **by** *blast*
from this obtain k_0 **where** $\langle f k_0 = k \rangle$ **by** *blast*
with *ticks-sub*[*OF assms*] h **have** $\langle \text{hamlet } ((\text{Rep-run sub}) k_0 c) \rangle$ **by** *blast*
with *k0prop* **have** $\langle k \in \text{image } f ?S \rangle$ **by** *blast*
}
thus $\langle ?R \subseteq \text{image } f ?S \rangle$ **by** *blast*
next
{ fix k **assume** $h:\langle k \in \text{image } f ?S \rangle$
from this obtain k_0 **where** $\langle f k_0 = k \wedge \text{hamlet } ((\text{Rep-run sub}) k_0 c) \rangle$ **by** *blast*
with *assms* **have** $\langle k \in ?R \rangle$ **using** *ticks-sub* **by** *blast*
}
thus $\langle \text{image } f ?S \subseteq ?R \rangle$ **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 $\langle (\text{LEAST } y. y \in f^{-1} S) = f (\text{LEAST } x. x \in S) \rangle$
using *Least-mono*[*OF strict-mono-mono, OF assms*] .

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

lemma *Least-nat-ex*:
 $\langle (n::\text{nat}) \in S \implies \exists x \in S. (\forall y \in S. x \leq y) \rangle$
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 $\langle \exists k::\text{nat}. \text{hamlet } ((\text{Rep-run sub}) k c) \rangle$
shows $\langle (\text{LEAST } k. k \in \{t. \text{hamlet } ((\text{Rep-run } r) t c)\}) = f (\text{LEAST } k. k \in \{t. \text{hamlet } ((\text{Rep-run sub}) t c)\}) \rangle$
 $\langle (\text{is } (\text{LEAST } k. k \in ?R) = f (\text{LEAST } k. k \in ?S)) \rangle$
proof –
from *assms*(2) **have** $\langle \exists x. x \in ?S \rangle$ **by** *simp*
hence *least*: $\langle \exists x \in ?S. \forall y \in ?S. x \leq y \rangle$
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**
 $\langle (\text{LEAST } y. y \in f^{-1} ?S) = f (\text{LEAST } x. x \in ?S) \rangle$.
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 $\langle \exists k. \text{hamlet } ((\text{Rep-run } r) k c) \rangle$
shows $\langle \exists k_0. \text{hamlet } ((\text{Rep-run sub}) k_0 c) \rangle$
proof –
from *assms*(2) **obtain** *k* **where** $\langle \text{hamlet } ((\text{Rep-run } r) k c) \rangle$ **by** *blast*
with *ticks-image-sub*[*OF assms*(1)] *ticks-sub*[*OF assms*(1)] **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 $\langle \text{hamlet } ((\text{Rep-run } r) k c) \rangle$
shows $\langle \exists k_0. f k_0 = k \wedge \text{hamlet } ((\text{Rep-run sub}) k_0 c) \rangle$
proof –
from *no-tick-sub*[*OF assms*(1)] *assms*(2) **have** $\langle \exists k_0. f k_0 = k \rangle$ **by** *blast*
from *this* **obtain** *k*₀ **where** $\langle f k_0 = k \rangle$ **by** *blast*

moreover with $\text{ticks-sub}[OF \text{ assms}(1)] \text{ assms}(2)$ have $\langle \text{hamlet } ((\text{Rep-run sub}) k_0 \ c) \rangle$ 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 $\langle \text{dilating } f \text{ sub } r \rangle$
 shows $\langle \{i. f \ m < i \wedge i < f \ n \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} \rangle$
 $= \text{image } f \ \langle \{i. m < i \wedge i < n \wedge \text{hamlet } ((\text{Rep-run sub}) \ i \ c)\} \rangle$
 (is $\langle ?RUN = \text{image } f \ ?SUB \rangle$)

proof

{ fix i assume $h: i \in ?SUB$
 hence $\langle m < i \wedge i < n \rangle$ by *simp*
 hence $\langle f \ m < f \ i \wedge f \ i < (f \ n) \rangle$ using *assms*
 by (*simp add: dilating-def dilating-fun-def strict-monoD strict-mono-less-eq*)
 moreover from h have $\langle \text{hamlet } ((\text{Rep-run sub}) \ i \ c) \rangle$ by *simp*
 hence $\langle \text{hamlet } ((\text{Rep-run } r) \ (f \ i) \ c) \rangle$ using $\text{ticks-sub}[OF \text{ assms}]$ by *blast*
 ultimately have $\langle f \ i \in ?RUN \rangle$ by *simp*
 } thus $\langle \text{image } f \ ?SUB \subseteq ?RUN \rangle$ by *blast*

next

{ fix i assume $h: i \in ?RUN$
 hence $\langle \text{hamlet } ((\text{Rep-run } r) \ i \ c) \rangle$ by *simp*
 from $\text{ticks-imp-ticks-subk}[OF \text{ assms this}]$
 obtain i_0 where $i_0 \text{prop}: f \ i_0 = i \wedge \text{hamlet } ((\text{Rep-run sub}) \ i_0 \ c) \rangle$ by *blast*
 with h have $\langle f \ m < f \ i_0 \wedge f \ i_0 < f \ n \rangle$ by *simp*
 moreover have $\langle \text{strict-mono } f \rangle$ using *assms dilating-def dilating-fun-def* by *blast*
 ultimately have $\langle m < i_0 \wedge i_0 < n \rangle$ using *strict-mono-less strict-mono-less-eq*
 by *blast*
 with $i_0 \text{prop}$ have $\langle \exists i_0. f \ i_0 = i \wedge i_0 \in ?SUB \rangle$ by *blast*
 } thus $\langle ?RUN \subseteq \text{image } f \ ?SUB \rangle$ by *blast*

qed

lemma *dilated-ticks-left:*

assumes $\langle \text{dilating } f \text{ sub } r \rangle$
 shows $\langle \{i. f \ m \leq i \wedge i < f \ n \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} \rangle$
 $= \text{image } f \ \langle \{i. m \leq i \wedge i < n \wedge \text{hamlet } ((\text{Rep-run sub}) \ i \ c)\} \rangle$
 (is $\langle ?RUN = \text{image } f \ ?SUB \rangle$)

proof

{ fix i assume $h: i \in ?SUB$
 hence $\langle m \leq i \wedge i < n \rangle$ by *simp*
 hence $\langle f \ m \leq f \ i \wedge f \ i < (f \ n) \rangle$ using *assms*
 by (*simp add: dilating-def dilating-fun-def strict-monoD strict-mono-less-eq*)
 moreover from h have $\langle \text{hamlet } ((\text{Rep-run sub}) \ i \ c) \rangle$ by *simp*
 hence $\langle \text{hamlet } ((\text{Rep-run } r) \ (f \ i) \ c) \rangle$ using $\text{ticks-sub}[OF \text{ assms}]$ by *blast*
 ultimately have $\langle f \ i \in ?RUN \rangle$ by *simp*
 } thus $\langle \text{image } f \ ?SUB \subseteq ?RUN \rangle$ by *blast*

next

{ **fix** i **assume** $h:(i \in ?RUN)$
 hence $\langle \text{hamlet } ((Rep\text{-run } r) \ i \ c) \rangle$ **by** *simp*
 from *ticks-imp-ticks-subk[OF assms this]*
 obtain i_0 **where** $i_0prop:\langle f \ i_0 = i \wedge \text{hamlet } ((Rep\text{-run } sub) \ i_0 \ c) \rangle$ **by** *blast*
 with h **have** $\langle f \ m \leq f \ i_0 \wedge f \ i_0 < f \ n \rangle$ **by** *simp*
 moreover **have** $\langle \text{strict-mono } f \rangle$ **using** *assms dilating-def dilating-fun-def* **by**
blast
 ultimately **have** $\langle m \leq i_0 \wedge i_0 < n \rangle$ **using** *strict-mono-less strict-mono-less-eq*
by *blast*
 with i_0prop **have** $\langle \exists i_0. f \ i_0 = i \wedge i_0 \in ?SUB \rangle$ **by** *blast*
 } **thus** $\langle ?RUN \subseteq \text{image } f \ ?SUB \rangle$ **by** *blast*
qed

lemma *dilated-ticks-right:*

assumes $\langle \text{dilating } f \ \text{sub } r \rangle$
shows $\langle \{i. f \ m < i \wedge i \leq f \ n \wedge \text{hamlet } ((Rep\text{-run } r) \ i \ c)\} \rangle$
 $= \text{image } f \ \langle \{i. m < i \wedge i \leq n \wedge \text{hamlet } ((Rep\text{-run } sub) \ i \ c)\} \rangle$
 (is $\langle ?RUN = \text{image } f \ ?SUB \rangle$)

proof

{ **fix** i **assume** $h:(i \in ?SUB)$
 hence $\langle m < i \wedge i \leq n \rangle$ **by** *simp*
 hence $\langle f \ m < f \ i \wedge f \ i \leq (f \ n) \rangle$ **using** *assms*
 by (*simp add: dilating-def dilating-fun-def strict-monoD strict-mono-less-eq*)
 moreover **from** h **have** $\langle \text{hamlet } ((Rep\text{-run } sub) \ i \ c) \rangle$ **by** *simp*
 hence $\langle \text{hamlet } ((Rep\text{-run } r) \ (f \ i) \ c) \rangle$ **using** *ticks-sub[OF assms]* **by** *blast*
 ultimately **have** $\langle f \ i \in ?RUN \rangle$ **by** *simp*
 } **thus** $\langle \text{image } f \ ?SUB \subseteq ?RUN \rangle$ **by** *blast*

next

{ **fix** i **assume** $h:(i \in ?RUN)$
 hence $\langle \text{hamlet } ((Rep\text{-run } r) \ i \ c) \rangle$ **by** *simp*
 from *ticks-imp-ticks-subk[OF assms this]*
 obtain i_0 **where** $i_0prop:\langle f \ i_0 = i \wedge \text{hamlet } ((Rep\text{-run } sub) \ i_0 \ c) \rangle$ **by** *blast*
 with h **have** $\langle f \ m < f \ i_0 \wedge f \ i_0 \leq f \ n \rangle$ **by** *simp*
 moreover **have** $\langle \text{strict-mono } f \rangle$ **using** *assms dilating-def dilating-fun-def* **by**
blast
 ultimately **have** $\langle m < i_0 \wedge i_0 \leq n \rangle$ **using** *strict-mono-less strict-mono-less-eq*
by *blast*
 with i_0prop **have** $\langle \exists i_0. f \ i_0 = i \wedge i_0 \in ?SUB \rangle$ **by** *blast*
 } **thus** $\langle ?RUN \subseteq \text{image } f \ ?SUB \rangle$ **by** *blast*

qed

lemma *dilated-ticks:*

assumes $\langle \text{dilating } f \ \text{sub } r \rangle$
shows $\langle \{i. f \ m \leq i \wedge i \leq f \ n \wedge \text{hamlet } ((Rep\text{-run } r) \ i \ c)\} \rangle$
 $= \text{image } f \ \langle \{i. m \leq i \wedge i \leq n \wedge \text{hamlet } ((Rep\text{-run } sub) \ i \ c)\} \rangle$
 (is $\langle ?RUN = \text{image } f \ ?SUB \rangle$)

proof

{ **fix** i **assume** $h:(i \in ?SUB)$
 hence $\langle m \leq i \wedge i \leq n \rangle$ **by** *simp*

hence $\langle f\ m \leq f\ i \wedge f\ i \leq (f\ n) \rangle$
 using *assms* by (*simp add: dilating-def dilating-fun-def strict-mono-less-eq*)
 moreover from *h* have $\langle \text{hamlet } ((\text{Rep-run } \text{sub})\ i\ c) \rangle$ by *simp*
 hence $\langle \text{hamlet } ((\text{Rep-run } r)\ (f\ i)\ c) \rangle$ using *ticks-sub[OF assms]* by *blast*
 ultimately have $\langle f\ i \in ?\text{RUN} \rangle$ by *simp*
 } thus $\langle \text{image } f\ ?\text{SUB} \subseteq ?\text{RUN} \rangle$ by *blast*
 next
 { fix *i* assume $h:\langle i \in ?\text{RUN} \rangle$
 hence $\langle \text{hamlet } ((\text{Rep-run } r)\ i\ c) \rangle$ by *simp*
 from *ticks-imp-ticks-subk[OF assms this]*
 obtain i_0 where $i_0\text{prop}:\langle f\ i_0 = i \wedge \text{hamlet } ((\text{Rep-run } \text{sub})\ i_0\ c) \rangle$ by *blast*
 with *h* have $\langle f\ m \leq f\ i_0 \wedge f\ i_0 \leq f\ n \rangle$ by *simp*
 moreover have $\langle \text{strict-mono } f \rangle$ using *assms dilating-def dilating-fun-def* by
blast
 ultimately have $\langle m \leq i_0 \wedge i_0 \leq n \rangle$ using *strict-mono-less-eq* by *blast*
 with $i_0\text{prop}$ have $\langle \exists i_0. f\ i_0 = i \wedge i_0 \in ?\text{SUB} \rangle$ by *blast*
 } thus $\langle ?\text{RUN} \subseteq \text{image } f\ ?\text{SUB} \rangle$ 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 $\langle \text{dilating } f\ \text{sub } r \rangle$
 and $\langle n < f\ 0 \rangle$
 shows $\langle \neg \text{hamlet } ((\text{Rep-run } r)\ n\ c) \rangle$
 proof –
 from *assms* have *False* by (*simp add: dilating-def dilating-fun-def*)
 thus *?thesis* ..
 qed

corollary *empty-dilated-prefix'*:

assumes $\langle \text{dilating } f\ \text{sub } r \rangle$
 shows $\langle \{i. f\ 0 \leq i \wedge i \leq f\ n \wedge \text{hamlet } ((\text{Rep-run } r)\ i\ c)\} = \{i. i \leq f\ n \wedge \text{hamlet } ((\text{Rep-run } r)\ i\ c)\} \rangle$
 proof –
 from *assms* have $\langle \text{strict-mono } f \rangle$ by (*simp add: dilating-def dilating-fun-def*)
 hence $\langle f\ 0 \leq f\ n \rangle$ unfolding *strict-mono-def* by (*simp add: less-mono-imp-le-mono*)
 hence $\langle \forall i. i \leq f\ n = (i < f\ 0) \vee (f\ 0 \leq i \wedge i \leq f\ n) \rangle$ by *auto*
 hence $\langle \{i. i \leq f\ n \wedge \text{hamlet } ((\text{Rep-run } r)\ i\ c)\} = \{i. i < f\ 0 \wedge \text{hamlet } ((\text{Rep-run } r)\ i\ c)\} \cup \{i. f\ 0 \leq i \wedge i \leq f\ n \wedge \text{hamlet } ((\text{Rep-run } r)\ i\ c)\} \rangle$
 by *auto*
 also have $\langle \dots = \{i. f\ 0 \leq i \wedge i \leq f\ n \wedge \text{hamlet } ((\text{Rep-run } r)\ i\ c)\} \rangle$
 using *empty-dilated-prefix[OF assms]* by *blast*
 finally show *?thesis* by *simp*
 qed

corollary *dilated-prefix*:

assumes $\langle \text{dilating } f\ \text{sub } r \rangle$

shows $\langle \{i. i \leq f\ n \wedge \text{hamlet } ((\text{Rep-run } r) i\ c)\} \rangle$
 $= \text{image } f\ \langle \{i. i \leq n \wedge \text{hamlet } ((\text{Rep-run sub}) i\ c)\} \rangle$
proof –
 have $\langle \{i. 0 \leq i \wedge i \leq f\ n \wedge \text{hamlet } ((\text{Rep-run } r) i\ c)\} \rangle$
 $= \text{image } f\ \langle \{i. 0 \leq i \wedge i \leq n \wedge \text{hamlet } ((\text{Rep-run sub}) i\ c)\} \rangle$
 using *dilated-ticks*[*OF assms*] *empty-dilated-prefix'*[*OF assms*] **by** *blast*
 thus *?thesis* **by** *simp*
qed

corollary *dilated-strict-prefix*:

assumes $\langle \text{dilating } f\ \text{sub } r \rangle$
 shows $\langle \{i. i < f\ n \wedge \text{hamlet } ((\text{Rep-run } r) i\ c)\} \rangle$
 $= \text{image } f\ \langle \{i. i < n \wedge \text{hamlet } ((\text{Rep-run sub}) i\ c)\} \rangle$
proof –
 from *assms* **have** *dil*: $\langle \text{dilating-fun } f\ r \rangle$ **unfolding** *dilating-def* **by** *simp*
 from *dil* **have** *f0*: $\langle f\ 0 = 0 \rangle$ **using** *dilating-fun-def* **by** *blast*
 from *dilating-fun-image-left*[*OF dil*, *of* $\langle 0 \rangle\ \langle n \rangle\ \langle c \rangle$]
 have $\langle \{i. f\ 0 \leq i \wedge i < f\ n \wedge \text{hamlet } ((\text{Rep-run } r) i\ c)\} \rangle$
 $= \text{image } f\ \langle \{i. 0 \leq i \wedge i < n \wedge \text{hamlet } ((\text{Rep-run } r) (f\ i)\ c)\} \rangle$.
 hence $\langle \{i. i < f\ n \wedge \text{hamlet } ((\text{Rep-run } r) i\ c)\} \rangle$
 $= \text{image } f\ \langle \{i. i < n \wedge \text{hamlet } ((\text{Rep-run } r) (f\ i)\ c)\} \rangle$
 using *f0* **by** *simp*
 also **have** $\langle \dots = \text{image } f\ \langle \{i. i < n \wedge \text{hamlet } ((\text{Rep-run sub}) i\ c)\} \rangle \rangle$
 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*:

$\langle \{i::\text{nat}. i = k \wedge P(i)\} \rangle = \langle \{i::\text{nat}. i = k \wedge P(k)\} \rangle$
by *auto*

The set definition and the function definition of *tick-count* are equivalent.

lemma *tick-count-is-fun*[*code*]: $\langle \text{tick-count } r\ c\ n = \text{run-tick-count } r\ c\ n \rangle$

proof (*induction n*)

case 0

have $\langle \text{tick-count } r\ c\ 0 = \text{card } \{i. i \leq 0 \wedge \text{hamlet } ((\text{Rep-run } r) i\ c)\} \rangle$
 by (*simp add: tick-count-def*)
 also **have** $\langle \dots = \text{card } \{i::\text{nat}. i = 0 \wedge \text{hamlet } ((\text{Rep-run } r) 0\ c)\} \rangle$
 using *le-zero-eq nat-sing-prop*[*of* $\langle 0 \rangle\ \langle \lambda i. \text{hamlet } ((\text{Rep-run } r) i\ c) \rangle$] **by** *simp*
 also **have** $\langle \dots = (\text{if } \text{hamlet } ((\text{Rep-run } r) 0\ c) \text{ then } 1 \text{ else } 0) \rangle$ **by** *simp*
 also **have** $\langle \dots = \text{run-tick-count } r\ c\ 0 \rangle$ **by** *simp*
 finally **show** *?case* .

next

case (*Suc k*)

show *?case*

proof (*cases* $\langle \text{hamlet } ((\text{Rep-run } r) (\text{Suc } k)\ c) \rangle$)

case *True*

hence $\langle \{i. i \leq \text{Suc } k \wedge \text{hamlet } ((\text{Rep-run } r) i\ c)\} \rangle = \text{insert } (\text{Suc } k)\ \langle \{i. i \leq$


```

 $k \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\rangle$ 
  by auto
  hence  $\langle \text{tick-count } r \ c \ (\text{Suc } k) = \text{Suc } (\text{tick-count } r \ c \ k) \rangle$ 
    by (simp add: tick-count-def)
  with Suc.IH have  $\langle \text{tick-count } r \ c \ (\text{Suc } k) = \text{Suc } (\text{run-tick-count } r \ c \ k) \rangle$  by
simp
  thus ?thesis by (simp add: True)
next
case False
  hence  $\langle \{i. i \leq \text{Suc } k \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} = \{i. i \leq k \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} \rangle$ 
    using le-Suc-eq by auto
  hence  $\langle \text{tick-count } r \ c \ (\text{Suc } k) = \text{tick-count } r \ c \ k \rangle$  by (simp add: tick-count-def)
  thus ?thesis using Suc.IH by (simp add: False)
qed
qed

```

The set definition and the function definition of *tick-count-strict* are equivalent.

lemma *tick-count-strict-suc*: $\langle \text{tick-count-strict } r \ c \ (\text{Suc } n) = \text{tick-count } r \ c \ n \rangle$
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 $\langle \text{tick-count-strict } r \ c \ n = 0 \rangle$ **unfolding** *tick-count-strict-def* **by** *simp*

also have $\langle \dots = \text{run-tick-count-strictly } r \ c \ 0 \rangle$ **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** $\langle n = \text{Suc } m \rangle$ **by** *blast*

hence $\langle \text{tick-count-strict } r \ c \ n = \text{tick-count } r \ c \ m \rangle$ **using** *tick-count-strict-suc* **by**

simp

also have $\langle \dots = \text{run-tick-count } r \ c \ m \rangle$ **using** *tick-count-is-fun*[*of* $\langle r \rangle \ \langle c \rangle \ \langle m \rangle$] .

also have $\langle \dots = \text{run-tick-count-strictly } r \ c \ (\text{Suc } m) \rangle$ **using** *run-tick-count-strictly.simps(2)[symmetric]*

.

finally show ?thesis using * by *simp*

qed

lemma *cong-suc-collect*:

assumes $\langle \bigwedge r \ K \ n. P \ r \ K \ n = P' \ r \ K \ n \rangle$

and $\langle \bigwedge r \ K \ n. Q \ r \ K \ n = Q' \ r \ K \ n \rangle$

and $\langle \bigwedge r \ K \ n. Q \ r \ K \ (\text{Suc } n) = P \ r \ K \ n \rangle$

shows $\langle \bigwedge K_1 \ K_2 \ n. \{r. P' \ r \ K_2 \ n \leq Q' \ r \ K_1 \ n\} = \{r. Q' \ r \ K_2 \ (\text{Suc } n) \leq Q' \ r \ K_1 \ n\} \rangle$

using *assms* **by** *auto*

lemma *strictly-precedes-alt-def1*:

$\langle \{ \varrho. \forall n::\text{nat}. (\text{run-tick-count } \varrho \ K_2 \ n) \leq (\text{run-tick-count-strictly } \varrho \ K_1 \ n) \} \rangle$
 $= \langle \{ \varrho. \forall n::\text{nat}. (\text{run-tick-count-strictly } \varrho \ K_2 \ (\text{Suc } n)) \leq (\text{run-tick-count-strictly } \varrho \ K_1 \ n) \} \rangle$
using *cong-suc-collect*[*of tick-count run-tick-count tick-count-strict run-tick-count-strictly,*
OF tick-count-is-fun tick-count-strict-is-fun tick-count-strict-suc]
by *simp*

lemma *zero-gt-all*:

assumes $\langle P \ (0::\text{nat}) \rangle$
and $\langle \bigwedge n. n > 0 \implies P \ n \rangle$
shows $\langle P \ n \rangle$
using *assms neq0-conv* **by** *blast*

lemma *strictly-precedes-alt-def2*:

$\langle \{ \varrho. \forall n::\text{nat}. (\text{run-tick-count } \varrho \ K_2 \ n) \leq (\text{run-tick-count-strictly } \varrho \ K_1 \ n) \} \rangle$
 $= \langle \{ \varrho. (\neg \text{hamlet } ((\text{Rep-run } \varrho) \ 0 \ K_2)) \wedge (\forall n::\text{nat}. (\text{run-tick-count } \varrho \ K_2 \ (\text{Suc } n))) \leq (\text{run-tick-count } \varrho \ K_1 \ n)) \} \rangle$
 $\langle \text{is } \langle ?P = ?P' \rangle \rangle$

proof

$\{ \text{fix } r::\langle 'a \text{ run} \rangle$
assume $\langle r \in ?P \rangle$
hence $\langle \forall n::\text{nat}. (\text{run-tick-count } r \ K_2 \ n) \leq (\text{run-tick-count-strictly } r \ K_1 \ n) \rangle$ **by**
simp
hence $1::\langle \forall n::\text{nat}. (\text{tick-count } r \ K_2 \ n) \leq (\text{tick-count-strict } r \ K_1 \ n) \rangle$
using *tick-count-is-fun*[*symmetric, of r*] *tick-count-strict-is-fun*[*symmetric, of*
r] **by** *simp*
hence $\langle \forall n::\text{nat}. (\text{tick-count-strict } r \ K_2 \ (\text{Suc } n)) \leq (\text{tick-count-strict } r \ K_1 \ n) \rangle$
using *tick-count-strict-suc*[*symmetric, of <r> <K2>*] **by** *simp*
hence $\langle \forall n::\text{nat}. (\text{tick-count-strict } r \ K_2 \ (\text{Suc } (\text{Suc } n))) \leq (\text{tick-count-strict } r \ K_1 \ (\text{Suc } n)) \rangle$ **by** *simp*
hence $\langle \forall n::\text{nat}. (\text{tick-count } r \ K_2 \ (\text{Suc } n)) \leq (\text{tick-count } r \ K_1 \ n) \rangle$
using *tick-count-strict-suc*[*symmetric, of <r>*] **by** *simp*
hence $\ast::\langle \forall n::\text{nat}. (\text{run-tick-count } r \ K_2 \ (\text{Suc } n)) \leq (\text{run-tick-count } r \ K_1 \ n) \rangle$
by (*simp add: tick-count-is-fun*)
from 1 **have** $\langle \text{tick-count } r \ K_2 \ 0 \leq \text{tick-count-strict } r \ K_1 \ 0 \rangle$ **by** *simp*
moreover **have** $\langle \text{tick-count-strict } r \ K_1 \ 0 = 0 \rangle$ **unfolding** *tick-count-strict-def*
by *simp*
ultimately **have** $\langle \text{tick-count } r \ K_2 \ 0 = 0 \rangle$ **by** *simp*
hence $\langle \neg \text{hamlet } ((\text{Rep-run } r) \ 0 \ K_2) \rangle$ **unfolding** *tick-count-def* **by** *auto*
with \ast **have** $\langle r \in ?P' \rangle$ **by** *simp*
 $\}$ **thus** $\langle ?P \subseteq ?P' \rangle$ **..**
 $\{ \text{fix } r::\langle 'a \text{ run} \rangle$
assume $h::\langle r \in ?P' \rangle$
hence $\langle \forall n::\text{nat}. (\text{run-tick-count } r \ K_2 \ (\text{Suc } n)) \leq (\text{run-tick-count } r \ K_1 \ n) \rangle$ **by**
simp
hence $\langle \forall n::\text{nat}. (\text{tick-count } r \ K_2 \ (\text{Suc } n)) \leq (\text{tick-count } r \ K_1 \ n) \rangle$
by (*simp add: tick-count-is-fun*)

hence $\langle \forall n::\text{nat}. (\text{tick-count } r \ K_2 \ (\text{Suc } n)) \leq (\text{tick-count-strict } r \ K_1 \ (\text{Suc } n)) \rangle$
 using *tick-count-strict-suc*[*symmetric*, of $\langle r \rangle \ K_1$] **by** *simp*
 hence $\ast: \langle \forall n. n > 0 \longrightarrow (\text{tick-count } r \ K_2 \ n) \leq (\text{tick-count-strict } r \ K_1 \ n) \rangle$
 using *gr0-implies-Suc* **by** *blast*
 have $\langle \text{tick-count-strict } r \ K_1 \ 0 = 0 \rangle$ **unfolding** *tick-count-strict-def* **by** *simp*
 moreover **from** h have $\langle \neg \text{hamlet } ((\text{Rep-run } r) \ 0 \ K_2) \rangle$ **by** *simp*
 hence $\langle \text{tick-count } r \ K_2 \ 0 = 0 \rangle$ **unfolding** *tick-count-def* **by** *auto*
 ultimately have $\langle \text{tick-count } r \ K_2 \ 0 \leq \text{tick-count-strict } r \ K_1 \ 0 \rangle$ **by** *simp*
 from *zero-gt-all*[of $\langle \lambda n. \text{tick-count } r \ K_2 \ n \leq \text{tick-count-strict } r \ K_1 \ n \rangle$, OF *this*
] \ast
 have $\langle \forall n. (\text{tick-count } r \ K_2 \ n) \leq (\text{tick-count-strict } r \ K_1 \ n) \rangle$ **by** *simp*
 hence $\langle \forall n. (\text{run-tick-count } r \ K_2 \ n) \leq (\text{run-tick-count-strictly } r \ K_1 \ n) \rangle$
by (*simp add: tick-count-is-fun tick-count-strict-is-fun*)
 hence $\langle r \in ?P \rangle \dots$
 } thus $\langle ?P' \subseteq ?P \rangle \dots$
qed

lemma *run-tick-count-suc*:

$\langle \text{run-tick-count } r \ c \ (\text{Suc } n) = (\text{if hamlet } ((\text{Rep-run } r) \ (\text{Suc } n) \ c)$
 $\text{then } \text{Suc } (\text{run-tick-count } r \ c \ n)$
 $\text{else } \text{run-tick-count } r \ c \ n) \rangle$

by *simp*

corollary *tick-count-suc*:

$\langle \text{tick-count } r \ c \ (\text{Suc } n) = (\text{if hamlet } ((\text{Rep-run } r) \ (\text{Suc } n) \ c)$
 $\text{then } \text{Suc } (\text{tick-count } r \ c \ n)$
 $\text{else } \text{tick-count } r \ c \ n) \rangle$

by (*simp add: tick-count-is-fun*)

lemma *card-suc*: $\langle \text{card } \{i. i \leq (\text{Suc } n) \wedge P \ i\} = \text{card } \{i. i \leq n \wedge P \ i\} + \text{card } \{i. i = (\text{Suc } n) \wedge P \ i\} \rangle$

proof –

have $\langle \{i. i \leq n \wedge P \ i\} \cap \{i. i = (\text{Suc } n) \wedge P \ i\} = \{\} \rangle$ **by** *auto*
 moreover have $\langle \{i. i \leq n \wedge P \ i\} \cup \{i. i = (\text{Suc } n) \wedge P \ i\} = \{i. i \leq (\text{Suc } n) \wedge P \ i\} \rangle$ **by** *auto*
 moreover have $\langle \text{finite } \{i. i \leq n \wedge P \ i\} \rangle$ **by** *simp*
 moreover have $\langle \text{finite } \{i. i = (\text{Suc } n) \wedge P \ i\} \rangle$ **by** *simp*
 ultimately show ?thesis **using** *card-Un-disjoint*[of $\langle \{i. i \leq n \wedge P \ i\} \rangle \langle \{i. i = \text{Suc } n \wedge P \ i\} \rangle$] **by** *simp*
qed

lemma *card-le-leq*:

assumes $\langle m < n \rangle$

shows $\langle \text{card } \{i::\text{nat}. m < i \wedge i \leq n \wedge P \ i\} = \text{card } \{i. m < i \wedge i < n \wedge P \ i\}$
 $+ \text{card } \{i. i = n \wedge P \ i\} \rangle$

proof –

have $\langle \{i::\text{nat}. m < i \wedge i < n \wedge P \ i\} \cap \{i. i = n \wedge P \ i\} = \{\} \rangle$ **by** *auto*
 moreover **with** *assms* have $\langle \{i::\text{nat}. m < i \wedge i < n \wedge P \ i\} \cup \{i. i = n \wedge P \ i\} = \{i. m < i \wedge i \leq n \wedge P \ i\} \rangle$ **by** *auto*

moreover have $\langle \text{finite } \{i. m < i \wedge i < n \wedge P i\} \rangle$ by *simp*
 moreover have $\langle \text{finite } \{i. i = n \wedge P i\} \rangle$ by *simp*
 ultimately show *?thesis* using *card-Un-disjoint*[of $\langle \{i. m < i \wedge i < n \wedge P i\} \rangle$
 $\langle \{i. i = n \wedge P i\} \rangle$] by *simp*
 qed

lemma *card-le-leq-0*: $\langle \text{card } \{i::\text{nat}. i \leq n \wedge P i\} = \text{card } \{i. i < n \wedge P i\} + \text{card } \{i. i = n \wedge P i\} \rangle$
proof –
 have $\langle \{i::\text{nat}. i < n \wedge P i\} \cap \{i. i = n \wedge P i\} = \{\} \rangle$ by *auto*
 moreover have $\langle \{i. i < n \wedge P i\} \cup \{i. i = n \wedge P i\} = \{i. i \leq n \wedge P i\} \rangle$ by *auto*
 moreover have $\langle \text{finite } \{i. i < n \wedge P i\} \rangle$ by *simp*
 moreover have $\langle \text{finite } \{i. i = n \wedge P i\} \rangle$ by *simp*
 ultimately show *?thesis* using *card-Un-disjoint*[of $\langle \{i. i < n \wedge P i\} \rangle$ $\langle \{i. i = n \wedge P i\} \rangle$] by *simp*
 qed

lemma *card-mnm*:
 assumes $\langle m < n \rangle$
 shows $\langle \text{card } \{i::\text{nat}. i < n \wedge P i\} = \text{card } \{i. i \leq m \wedge P i\} + \text{card } \{i. m < i \wedge i < n \wedge P i\} \rangle$
proof –
 have $1: \langle \{i::\text{nat}. i \leq m \wedge P i\} \cap \{i. m < i \wedge i < n \wedge P i\} = \{\} \rangle$ by *auto*
 from *assms* have $\langle \forall i::\text{nat}. i < n = (i \leq m) \vee (m < i \wedge i < n) \rangle$ using *less-trans*
 by *auto*
 hence 2:
 $\langle \{i::\text{nat}. i < n \wedge P i\} = \{i. i \leq m \wedge P i\} \cup \{i. m < i \wedge i < n \wedge P i\} \rangle$ by *blast*
 have 3: $\langle \text{finite } \{i. i \leq m \wedge P i\} \rangle$ by *simp*
 have 4: $\langle \text{finite } \{i. m < i \wedge i < n \wedge P i\} \rangle$ by *simp*
 from *card-Un-disjoint*[OF 3 4 1] 2 show *?thesis* by *simp*
 qed

lemma *card-mnm'*:
 assumes $\langle m < n \rangle$
 shows $\langle \text{card } \{i::\text{nat}. i < n \wedge P i\} = \text{card } \{i. i < m \wedge P i\} + \text{card } \{i. m \leq i \wedge i < n \wedge P i\} \rangle$
proof –
 have $1: \langle \{i::\text{nat}. i < m \wedge P i\} \cap \{i. m \leq i \wedge i < n \wedge P i\} = \{\} \rangle$ by *auto*
 from *assms* have $\langle \forall i::\text{nat}. i < n = (i < m) \vee (m \leq i \wedge i < n) \rangle$ using *less-trans*
 by *auto*
 hence 2:
 $\langle \{i::\text{nat}. i < n \wedge P i\} = \{i. i < m \wedge P i\} \cup \{i. m \leq i \wedge i < n \wedge P i\} \rangle$ by *blast*
 have 3: $\langle \text{finite } \{i. i < m \wedge P i\} \rangle$ by *simp*
 have 4: $\langle \text{finite } \{i. m \leq i \wedge i < n \wedge P i\} \rangle$ by *simp*
 from *card-Un-disjoint*[OF 3 4 1] 2 show *?thesis* by *simp*
 qed

lemma *nat-interval-union*:

assumes $\langle m \leq n \rangle$
shows $\langle \{i::nat. i \leq n \wedge P\ i\} = \{i::nat. i \leq m \wedge P\ i\} \cup \{i::nat. m < i \wedge i \leq n \wedge P\ i\} \rangle$
using *assms le-cases nat-less-le* **by** *auto*

lemma *no-tick-before-suc*:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$
and $\langle f\ n < k \wedge k < f\ (Suc\ n) \rangle$
shows $\langle \neg \text{hamlet } ((Rep-run\ r)\ k\ c) \rangle$
proof –
from *assms(1)* **have** $\text{smf}::\langle \text{strict-mono } f \rangle$ **by** (*simp add: dilating-def dilating-fun-def*)
{ fix k **assume** $h::\langle f\ n < k \wedge k < f\ (Suc\ n) \wedge \text{hamlet } ((Rep-run\ r)\ k\ c) \rangle$
hence $\langle \exists k_0. f\ k_0 = k \rangle$ **using** *assms(1) dilating-def dilating-fun-def* **by** *blast*
from this **obtain** k_0 **where** $\langle f\ k_0 = k \rangle$ **by** *blast*
with h **have** $\langle f\ n < f\ k_0 \wedge f\ k_0 < f\ (Suc\ n) \rangle$ **by** *simp*
hence *False* **using** *smf not-less-eq strict-mono-less* **by** *blast*
} **thus** *?thesis* **using** *assms(2)* **by** *blast*
qed

lemma *tick-count-fsuc*:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$
shows $\langle \text{tick-count } r\ c\ (f\ (Suc\ n)) = \text{tick-count } r\ c\ (f\ n) + \text{card } \{k. k = f\ (Suc\ n) \wedge \text{hamlet } ((Rep-run\ r)\ k\ c)\} \rangle$
proof –
have $\text{smf}::\langle \text{strict-mono } f \rangle$ **using** *assms dilating-def dilating-fun-def* **by** *blast*
moreover **have** $\langle \text{finite } \{k. k \leq f\ n \wedge \text{hamlet } ((Rep-run\ r)\ k\ c)\} \rangle$ **by** *simp*
moreover **have** $\langle \text{finite } \{k. f\ n < k \wedge k \leq f\ (Suc\ n) \wedge \text{hamlet } ((Rep-run\ r)\ k\ c)\} \rangle$ **by** *simp*
ultimately **have** $\langle \{k. k \leq f\ (Suc\ n) \wedge \text{hamlet } ((Rep-run\ r)\ k\ c)\} = \{k. k \leq f\ n \wedge \text{hamlet } ((Rep-run\ r)\ k\ c)\} \cup \{k. f\ n < k \wedge k \leq f\ (Suc\ n) \wedge \text{hamlet } ((Rep-run\ r)\ k\ c)\} \rangle$
by (*simp add: nat-interval-union strict-mono-less-eq*)
moreover **have** $\langle \{k. k \leq f\ n \wedge \text{hamlet } ((Rep-run\ r)\ k\ c)\} \cap \{k. f\ n < k \wedge k \leq f\ (Suc\ n) \wedge \text{hamlet } ((Rep-run\ r)\ k\ c)\} = \{\} \rangle$
by *auto*
ultimately **have** $\langle \text{card } \{k. k \leq f\ (Suc\ n) \wedge \text{hamlet } (Rep-run\ r\ k\ c)\} = \text{card } \{k. k \leq f\ n \wedge \text{hamlet } (Rep-run\ r\ k\ c)\} + \text{card } \{k. f\ n < k \wedge k \leq f\ (Suc\ n) \wedge \text{hamlet } (Rep-run\ r\ k\ c)\} \rangle$
by (*simp add: * card-Un-disjoint*)
moreover **from** *no-tick-before-suc[OF assms]* **have**
 $\langle \{k. f\ n < k \wedge k \leq f\ (Suc\ n) \wedge \text{hamlet } ((Rep-run\ r)\ k\ c)\} = \{k. k = f\ (Suc\ n) \wedge \text{hamlet } ((Rep-run\ r)\ k\ c)\} \rangle$
using *smf strict-mono-less* **by** *fastforce*
ultimately **show** *?thesis* **by** (*simp add: tick-count-def*)
qed

lemma *card-sing-prop*: $\langle \text{card } \{i. i = n \wedge P\ i\} = (\text{if } P\ n \text{ then } 1 \text{ else } 0) \rangle$

proof $\langle \text{cases } \langle P\ n \rangle$

case *True*

hence $\langle \{i. i = n \wedge P\ i\} = \{n\} \rangle$ **by** $\langle \text{simp add: Collect-conv-if} \rangle$

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

next

case *False*

hence $\langle \{i. i = n \wedge P\ i\} = \{\} \rangle$ **by** $\langle \text{simp add: Collect-conv-if} \rangle$

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

qed

corollary *tick-count-f-suc*:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$

shows $\langle \text{tick-count } r\ c\ (f\ (\text{Suc } n)) = \text{tick-count } r\ c\ (f\ n) + (\text{if hamlet } ((\text{Rep-run } r)\ (f\ (\text{Suc } n))\ c) \text{ then } 1 \text{ else } 0) \rangle$

using *tick-count-fsuc*[*OF* *assms*] *card-sing-prop*[*of* $\langle f\ (\text{Suc } n) \rangle \langle \lambda k. \text{hamlet } ((\text{Rep-run } r)\ k\ c) \rangle$] **by** *simp*

corollary *tick-count-f-suc-suc*:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$

shows $\langle \text{tick-count } r\ c\ (f\ (\text{Suc } n)) = (\text{if hamlet } ((\text{Rep-run } r)\ (f\ (\text{Suc } n))\ c) \text{ then } \text{Suc } (\text{tick-count } r\ c\ (f\ n)) \text{ else } \text{tick-count } r\ c\ (f\ n)) \rangle$

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 $\langle \text{tick-count } r\ c\ (f\ (\text{Suc } n)) = (\text{if hamlet } ((\text{Rep-run } \text{sub})\ (\text{Suc } n)\ c) \text{ then } \text{Suc } (\text{tick-count } r\ c\ (f\ n)) \text{ else } \text{tick-count } r\ c\ (f\ n)) \rangle$

using *tick-count-f-suc-suc*[*OF* *assms*] *assms* **by** $\langle \text{simp add: dilating-def} \rangle$

lemma *tick-count-sub*:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$

shows $\langle \text{tick-count } \text{sub } c\ n = \text{tick-count } r\ c\ (f\ n) \rangle$

proof –

have $\langle \text{tick-count } \text{sub } c\ n = \text{card } \{i. i \leq n \wedge \text{hamlet } ((\text{Rep-run } \text{sub})\ i\ c) \} \rangle$

using *tick-count-def*[*of* $\langle \text{sub} \rangle \langle c \rangle \langle n \rangle$] .

also have $\langle \dots = \text{card } (\text{image } f\ \{i. i \leq n \wedge \text{hamlet } ((\text{Rep-run } \text{sub})\ i\ c) \}) \rangle$

using *assms* *dilating-def* *dilating-injects*[*OF* *assms*] **by** $\langle \text{simp add: card-image} \rangle$

also have $\langle \dots = \text{card } \{i. i \leq f\ n \wedge \text{hamlet } ((\text{Rep-run } r)\ i\ c) \} \rangle$

using *dilated-prefix*[*OF* *assms*, *symmetric*, *of* $\langle n \rangle \langle c \rangle$] **by** *simp*

also have $\langle \dots = \text{tick-count } r\ c\ (f\ n) \rangle$

using *tick-count-def*[*of* $\langle r \rangle \langle c \rangle \langle f\ n \rangle$] **by** *simp*

finally show *?thesis* .

qed

corollary *run-tick-count-sub*:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$

shows $\langle \text{run-tick-count sub } c \ n = \text{run-tick-count } r \ c \ (f \ n) \rangle$
proof –
have $\langle \text{run-tick-count sub } c \ n = \text{tick-count sub } c \ n \rangle$
using $\text{tick-count-is-fun}[\text{of } \langle \text{sub} \rangle \ c \ n, \text{symmetric}]$.
also from $\text{tick-count-sub}[OF \ \text{assms}]$ **have** $\langle \dots = \text{tick-count } r \ c \ (f \ n) \rangle$.
also have $\langle \dots = \#_{\leq} r \ c \ (f \ n) \rangle$ **using** $\text{tick-count-is-fun}[\text{of } r \ c \ (f \ n)]$.
finally show $?thesis$.
qed

lemma *tick-count-strict-0*:
assumes $\langle \text{dilating } f \ \text{sub } r \rangle$
shows $\langle \text{tick-count-strict } r \ c \ (f \ 0) = 0 \rangle$
proof –
from assms **have** $\langle f \ 0 = 0 \rangle$ **by** $(\text{simp add: dilating-def dilating-fun-def})$
thus $?thesis$ **unfolding** *tick-count-strict-def* **by** simp
qed

lemma *tick-count-latest*:
assumes $\langle \text{dilating } f \ \text{sub } r \rangle$
and $\langle f \ n_p < n \wedge (\forall k. f \ n_p < k \wedge k \leq n \longrightarrow (\# k_0. f \ k_0 = k)) \rangle$
shows $\langle \text{tick-count } r \ c \ n = \text{tick-count } r \ c \ (f \ n_p) \rangle$
proof –
have $\text{union}:\langle \{i. i \leq n \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} =$
 $\{i. i \leq f \ n_p \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\}$
 $\cup \{i. f \ n_p < i \wedge i \leq n \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} \rangle$ **using** $\text{assms}(2)$ **by**
auto
have $\text{partition}:\langle \{i. i \leq f \ n_p \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\}$
 $\cap \{i. f \ n_p < i \wedge i \leq n \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} = \{\} \rangle$
by $(\text{simp add: disjoint-iff-not-equal})$
from assms **have** $\langle \{i. f \ n_p < i \wedge i \leq n \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} = \{\} \rangle$
using *no-tick-sub* **by** *fastforce*
with union **and** partition **show** $?thesis$ **by** $(\text{simp add: tick-count-def})$
qed

lemma *tick-count-strict-stable*:
assumes $\langle \text{dilating } f \ \text{sub } r \rangle$
assumes $\langle f \ n < k \wedge k < (f \ (\text{Suc } n)) \rangle$
shows $\langle \text{tick-count-strict } r \ c \ k = \text{tick-count-strict } r \ c \ (f \ (\text{Suc } n)) \rangle$
proof –
from $\text{assms}(1)$ **have** $\text{smf}:\langle \text{strict-mono } f \rangle$ **by** $(\text{simp add: dilating-def dilating-fun-def})$
from $\text{assms}(2)$ **have** $\langle f \ n < k \rangle$ **by** simp
hence $\langle \forall i. k \leq i \longrightarrow f \ n < i \rangle$ **by** simp
with $\text{no-tick-before-suc}[OF \ \text{assms}(1)]$ **have**
 $\ast:\langle \forall i. k \leq i \wedge i < f \ (\text{Suc } n) \longrightarrow \neg \text{hamlet } ((\text{Rep-run } r) \ i \ c) \rangle$ **by** *blast*
from *tick-count-strict-def* **have** $\langle \text{tick-count-strict } r \ c \ (f \ (\text{Suc } n)) = \text{card } \{i. i <$
 $f \ (\text{Suc } n) \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} \rangle$.
also have $\langle \dots = \text{card } \{i. i < k \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} + \text{card } \{i. k \leq i \wedge$
 $i < f \ (\text{Suc } n) \wedge \text{hamlet } ((\text{Rep-run } r) \ i \ c)\} \rangle$
using *card-mnm'* $\text{assms}(2)$ **by** simp

also have $\langle \dots = \text{card } \{i. i < k \wedge \text{hamlet } ((\text{Rep-run } r) i c)\} \rangle$ **using** * **by** *simp*
 finally **show** ?thesis **by** (*simp add: tick-count-strict-def*)
qed

lemma *tick-count-strict-sub*:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$

shows $\langle \text{tick-count-strict sub } c n = \text{tick-count-strict } r c (f n) \rangle$

proof –

have $\langle \text{tick-count-strict sub } c n = \text{card } \{i. i < n \wedge \text{hamlet } ((\text{Rep-run sub}) i c)\} \rangle$

using *tick-count-strict-def*[*of* $\langle \text{sub} \rangle \langle c \rangle \langle n \rangle$] .

also have $\langle \dots = \text{card } (\text{image } f \{i. i < n \wedge \text{hamlet } ((\text{Rep-run sub}) i c)\}) \rangle$

using *assms dilating-def dilating-injects*[*OF assms*] **by** (*simp add: card-image*)

also have $\langle \dots = \text{card } \{i. i < f n \wedge \text{hamlet } ((\text{Rep-run } r) i c)\} \rangle$

using *dilated-strict-prefix*[*OF assms, symmetric, of* $\langle n \rangle \langle c \rangle$] **by** *simp*

also have $\langle \dots = \text{tick-count-strict } r c (f n) \rangle$

using *tick-count-strict-def*[*of* $\langle r \rangle \langle c \rangle \langle f n \rangle$] **by** *simp*

finally show ?thesis .

qed

lemma *card-prop-mono*:

assumes $\langle m \leq n \rangle$

shows $\langle \text{card } \{i::\text{nat}. i \leq m \wedge P i\} \leq \text{card } \{i. i \leq n \wedge P i\} \rangle$

proof –

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

moreover have $\langle \text{finite } \{i. i \leq n \wedge P i\} \rangle$ **by** *simp*

ultimately show ?thesis **by** (*simp add: card-mono*)

qed

lemma *mono-tick-count*:

$\langle \text{mono } (\lambda k. \text{tick-count } r c k) \rangle$

proof

{ **fix** $x y::\text{nat}$

assume $\langle x \leq y \rangle$

from *card-prop-mono*[*OF this*] **have** $\langle \text{tick-count } r c x \leq \text{tick-count } r c y \rangle$

unfolding *tick-count-def* **by** *simp*

} **thus** $\langle \bigwedge x y. x \leq y \implies \text{tick-count } r c x \leq \text{tick-count } r c y \rangle$.

qed

lemma *greatest-prev-image*:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$

shows $\langle (\nexists n_0. f n_0 = n) \implies (\exists n_p. f n_p < n \wedge (\forall k. f n_p < k \wedge k \leq n \longrightarrow (\nexists k_0. f k_0 = k))) \rangle$

proof (*induction n*)

case 0

with *assms* **have** $\langle f 0 = 0 \rangle$ **by** (*simp add: dilating-def dilating-fun-def*)

thus ?case **using** 0.premis **by** *blast*

next

case (*Suc n*)

show ?case


```

proof (cases  $\langle \exists n_0. f\ n_0 = n \rangle$ )
  case True
    from this obtain  $n_0$  where  $\langle f\ n_0 = n \rangle$  by blast
    hence  $\langle f\ n_0 < (Suc\ n) \wedge (\forall k. f\ n_0 < k \wedge k \leq (Suc\ n) \longrightarrow (\nexists k_0. f\ k_0 = k)) \rangle$ 
      using Suc.premys Suc-leI le-antisym by blast
    thus ?thesis by blast
  next
    case False
    from Suc.IH[OF this] obtain  $n_p$ 
      where  $\langle f\ n_p < n \wedge (\forall k. f\ n_p < k \wedge k \leq n \longrightarrow (\nexists k_0. f\ k_0 = k)) \rangle$  by blast
    hence  $\langle f\ n_p < Suc\ n \wedge (\forall k. f\ n_p < k \wedge k \leq n \longrightarrow (\nexists k_0. f\ k_0 = k)) \rangle$  by simp
    with Suc(2) have  $\langle f\ n_p < (Suc\ n) \wedge (\forall k. f\ n_p < k \wedge k \leq (Suc\ n) \longrightarrow (\nexists k_0. f\ k_0 = k)) \rangle$ 
      using le-Suc-eq by auto
    thus ?thesis by blast
qed
qed

```

```

lemma strict-mono-suc:
  assumes  $\langle \text{strict-mono}\ f \rangle$ 
  and  $\langle f\ sn = Suc\ (f\ n) \rangle$ 
  shows  $\langle sn = Suc\ n \rangle$ 
proof –
  from assms(2) have  $\langle f\ sn > f\ n \rangle$  by simp
  with strict-mono-less[OF assms(1)] have  $\langle sn > n \rangle$  by simp
  moreover have  $\langle sn \leq Suc\ n \rangle$ 
  proof –
    { assume  $\langle sn > Suc\ n \rangle$ 
      from this obtain  $i$  where  $\langle n < i \wedge i < sn \rangle$  by blast
      hence  $\langle f\ n < f\ i \wedge f\ i < f\ sn \rangle$  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

```

```

lemma next-non-stuttering:
  assumes  $\langle \text{dilating}\ f\ \text{sub}\ r \rangle$ 
  and  $\langle f\ n_p < n \wedge (\forall k. f\ n_p < k \wedge k \leq n \longrightarrow (\nexists k_0. f\ k_0 = k)) \rangle$ 
  and  $\langle f\ sn_0 = Suc\ n \rangle$ 
  shows  $\langle sn_0 = Suc\ n_p \rangle$ 
proof –
  from assms(1) have smf: $\langle \text{strict-mono}\ f \rangle$  by (simp add: dilating-def dilating-fun-def)
  from assms(2) have  $\langle \forall k. f\ n_p < k \wedge k < Suc\ n \longrightarrow (\nexists k_0. f\ k_0 = k) \rangle$  by simp
  from assms(2) have  $\langle f\ n_p < n \rangle$  by simp
  with smf assms(3) have  $\langle sn_0 > n_p \rangle$  using strict-mono-less by fastforce
  have  $\langle Suc\ n \leq f\ (Suc\ n_p) \rangle$ 
  proof –
    { assume  $\langle h: Suc\ n > f\ (Suc\ n_p) \rangle$ 

```

hence $\langle \text{Suc } n_p < sn_0 \rangle$ using $** \text{Suc-lessI } \text{assms}(3)$ by *fastforce*
 hence $\langle \exists k. k > n_p \wedge f k < \text{Suc } n \rangle$ using *h* by *blast*
 with $*$ have *False* using *smf strict-mono-less* by *blast*
 } thus ?thesis using *not-less* by *blast*
 qed
 hence $\langle sn_0 \leq \text{Suc } n_p \rangle$ using *assms(3) smf* using *strict-mono-less-eq* by *fastforce*
 with $**$ show ?thesis by *simp*
 qed

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 $*:\langle \text{dilating } f \text{ sub } r \rangle$ by *blast*
 show ?thesis
 proof (induction *n*)
 case 0
 from *assms(2)* have $\langle \text{run-tick-count sub } a \ 0 \leq \text{run-tick-count sub } b \ 0 \rangle$..
 with *run-tick-count-sub[OF *, of - 0]* have $\langle \text{run-tick-count } r \ a \ (f \ 0) \leq$
run-tick-count } r \ b \ (f \ 0) \rangle by *simp*
 moreover from $*$ have $\langle f \ 0 = 0 \rangle$ by (*simp add: dilating-def dilating-fun-def*)
 ultimately show ?case by *simp*
 next
 case $(\text{Suc } n')$ thus ?case
 proof (cases $\langle \exists n_0. f \ n_0 = \text{Suc } n' \rangle$)
 case True
 from *this* obtain n_0 where $fn0:\langle f \ n_0 = \text{Suc } n' \rangle$ by *blast*
 show ?thesis
 proof (cases $\langle \text{hamlet } ((\text{Rep-run sub}) \ n_0 \ a) \rangle$)
 case True
 have $\langle \text{run-tick-count } r \ a \ (f \ n_0) \leq \text{run-tick-count } r \ b \ (f \ n_0) \rangle$
 using *assms(2) run-tick-count-sub[OF *]* by *simp*
 thus ?thesis by (*simp add: fn0*)
 next
 case False
 hence $\langle \neg \text{hamlet } ((\text{Rep-run } r) \ (\text{Suc } n') \ a) \rangle$ using $* \ fn0$ ticks-sub by
fastforce
 thus ?thesis by (*simp add: Suc.IH le-SucI*)
 qed
 next
 case False
 thus ?thesis using $* \text{Suc.IH no-tick-sub}$ by *fastforce*
 qed
 qed
 qed

lemma *stutter-no-time*:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$

and $\langle \bigwedge k. f\ n < k \wedge k \leq m \implies (\nexists k_0. f\ k_0 = k) \rangle$
 and $\langle m > f\ n \rangle$
 shows $\langle \text{time } ((\text{Rep-run } r)\ m\ c) = \text{time } ((\text{Rep-run } r)\ (f\ n)\ c) \rangle$
proof –
 from *assms* have $\langle \forall k. k < m - (f\ n) \longrightarrow (\nexists k_0. f\ k_0 = \text{Suc } ((f\ n) + k)) \rangle$ **by**
simp
 hence $\langle \forall k. k < m - (f\ n) \longrightarrow \text{time } ((\text{Rep-run } r)\ (\text{Suc } ((f\ n) + k))\ c) = \text{time } ((\text{Rep-run } r)\ ((f\ n) + k)\ c) \rangle$
 using *assms*(1) **by** (*simp add: dilating-def dilating-fun-def*)
 hence $\langle \forall k. k < m - (f\ n) \longrightarrow \text{time } ((\text{Rep-run } r)\ (\text{Suc } ((f\ n) + k))\ c) = \text{time } ((\text{Rep-run } r)\ (f\ n)\ c) \rangle$
 using *bounded-suc-ind*[of $\langle m - (f\ n) \rangle$ $\langle \lambda k. \text{time } ((\text{Rep-run } r)\ k\ c) \rangle$ $\langle f\ n \rangle$] **by** *blast*
 from *assms*(3) **obtain** m_0 **where** $m_0 : \text{Suc } m_0 = m - (f\ n)$ **using** *Suc-diff-Suc*
by *blast*
 with $*$ **have** $\langle \text{time } ((\text{Rep-run } r)\ (\text{Suc } ((f\ n) + m_0))\ c) = \text{time } ((\text{Rep-run } r)\ (f\ n)\ c) \rangle$ **by** *auto*
 moreover from m_0 **have** $\langle \text{Suc } ((f\ n) + m_0) = m \rangle$ **by** *simp*
 ultimately **show** *?thesis* **by** *simp*
qed

lemma *time-stuttering*:

assumes $\langle \text{dilating } f\ \text{sub } r \rangle$
 and $\langle \text{time } ((\text{Rep-run } \text{sub})\ n\ c) = \tau \rangle$
 and $\langle \bigwedge k. f\ n < k \wedge k \leq m \implies (\nexists k_0. f\ k_0 = k) \rangle$
 and $\langle m > f\ n \rangle$
 shows $\langle \text{time } ((\text{Rep-run } r)\ m\ c) = \tau \rangle$
proof –
 from *assms*(3) **have** $\langle \text{time } ((\text{Rep-run } r)\ m\ c) = \text{time } ((\text{Rep-run } r)\ (f\ n)\ c) \rangle$
 using *stutter-no-time*[OF *assms*(1,3,4)] **by** *blast*
 also from *assms*(1,2) **have** $\langle \text{time } ((\text{Rep-run } r)\ (f\ n)\ c) = \tau \rangle$ **by** (*simp add: dilating-def*)
 finally **show** *?thesis* .
qed

lemma *first-time-image*:

assumes $\langle \text{dilating } f\ \text{sub } r \rangle$
 shows $\langle \text{first-time } \text{sub } c\ n\ t = \text{first-time } r\ c\ (f\ n)\ t \rangle$
proof
 assume $\langle \text{first-time } \text{sub } c\ n\ t \rangle$
 with *before-first-time*[OF *this*]
 have $\langle \text{time } ((\text{Rep-run } \text{sub})\ n\ c) = t \wedge (\forall m < n. \text{time } ((\text{Rep-run } \text{sub})\ m\ c) < t) \rangle$
 by (*simp add: first-time-def*)
 moreover **have** $\langle \forall n\ c. \text{time } ((\text{Rep-run } \text{sub})\ n\ c) = \text{time } ((\text{Rep-run } r)\ (f\ n)\ c) \rangle$
 using *assms*(1) **by** (*simp add: dilating-def*)
 ultimately **have** $\langle \text{time } ((\text{Rep-run } r)\ (f\ n)\ c) = t \wedge (\forall m < n. \text{time } ((\text{Rep-run } r)\ (f\ m)\ c) < t) \rangle$
 by *simp*

```

have  $\langle \forall m < f\ n. \text{time } ((\text{Rep-run } r) \ m \ c) < t \rangle$ 
proof -
{ fix m assume hyp:  $\langle m < f\ n \rangle$ 
  have  $\langle \text{time } ((\text{Rep-run } r) \ m \ c) < t \rangle$ 
  proof (cases  $\langle \exists m_0. f\ m_0 = m \rangle$ )
    case True
      from this obtain m0 where mm0:  $\langle m = f\ m_0 \rangle$  by blast
      with hyp have m0n:  $\langle m_0 < n \rangle$  using assms(1)
      by (simp add: dilating-def dilating-fun-def strict-mono-less)
      hence  $\langle \text{time } ((\text{Rep-run sub}) \ m_0 \ c) < t \rangle$  using * by blast
      thus ?thesis by (simp add: mm0 m0n **)
    next
      case False
        hence  $\langle \exists m_p. f\ m_p < m \wedge (\forall k. f\ m_p < k \wedge k \leq m \longrightarrow (\nexists k_0. f\ k_0 = k)) \rangle$ 
        using greatest-prev-image[OF assms] by simp
        from this obtain mp where mp:  $\langle f\ m_p < m \wedge (\forall k. f\ m_p < k \wedge k \leq m \longrightarrow (\nexists k_0. f\ k_0 = k)) \rangle$ 
        by blast
        hence  $\langle \text{time } ((\text{Rep-run } r) \ m \ c) = \text{time } ((\text{Rep-run sub}) \ m_p \ c) \rangle$ 
        using time-stuttering[OF assms] by blast
        also from hyp mp have  $\langle f\ m_p < f\ n \rangle$  by linarith
        hence  $\langle m_p < n \rangle$  using assms
        by (simp add: dilating-def dilating-fun-def strict-mono-less)
        hence  $\langle \text{time } ((\text{Rep-run sub}) \ m_p \ c) < t \rangle$  using * by simp
        finally show ?thesis by simp
      qed
    } thus ?thesis by simp
  qed
with ** show  $\langle \text{first-time } r \ c \ (f\ n) \ t \rangle$  by (simp add: alt-first-time-def)
next
  assume  $\langle \text{first-time } r \ c \ (f\ n) \ t \rangle$ 
  hence *:  $\langle \text{time } ((\text{Rep-run } r) \ (f\ n) \ c) = t \wedge (\forall k < f\ n. \text{time } ((\text{Rep-run } r) \ k \ c) < t) \rangle$ 
  by (simp add: first-time-def before-first-time)
  hence  $\langle \text{time } ((\text{Rep-run sub}) \ n \ c) = t \rangle$  using assms dilating-def by blast
  moreover from * have  $\langle (\forall k < n. \text{time } ((\text{Rep-run sub}) \ k \ c) < t) \rangle$ 
  using assms dilating-def dilating-fun-def strict-monoD by fastforce
  ultimately show  $\langle \text{first-time sub } c \ n \ t \rangle$  by (simp add: alt-first-time-def)
qed

lemma first-dilated-instant:
  assumes  $\langle \text{strict-mono } f \rangle$ 
  and  $\langle f \ (0::nat) = (0::nat) \rangle$ 
  shows  $\langle \text{Max } \{i. f\ i \leq 0\} = 0 \rangle$ 
proof -
  from assms(2) have  $\langle \forall n > 0. f\ n > 0 \rangle$  using strict-monoD[OF assms(1)] by
  force
  hence  $\langle \forall n \neq 0. \neg(f\ n \leq 0) \rangle$  by simp
  with assms(2) have  $\langle \{i. f\ i \leq 0\} = \{0\} \rangle$  by blast

```

thus ?thesis by simp
qed

lemma not-image-stut:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$

and $\langle n_0 = \text{Max } \{i. f i \leq n\} \rangle$

and $\langle f n_0 < k \wedge k \leq n \rangle$

shows $\langle \nexists k_0. f k_0 = k \rangle$

proof –

from assms(1) have smf: $\langle \text{strict-mono } f \rangle$

and fxge: $\langle \forall x. f x \geq x \rangle$

by (auto simp add: dilating-def dilating-fun-def)

have finite-prefix: $\langle \text{finite } \{i. f i \leq n\} \rangle$ by (simp add: finite-less-ub fxge)

from assms(1) have $\langle f 0 \leq n \rangle$ by (simp add: dilating-def dilating-fun-def)

hence $\langle \{i. f i \leq n\} \neq \{\} \rangle$ by blast

from assms(3) fxge have $\langle f n_0 < n \rangle$ by linarith

from assms(2) have $\langle \forall x > n_0. f x > n \rangle$ using Max.coboundedI[OF finite-prefix]

using not-le by auto

with assms(3) strict-mono-less[OF smf] show ?thesis by auto

qed

lemma contracting-inverse:

assumes $\langle \text{dilating } f \text{ sub } r \rangle$

shows $\langle \text{contracting } (\text{dil-inverse } f) \text{ } r \text{ sub } f \rangle$

proof –

from assms have smf: $\langle \text{strict-mono } f \rangle$

and no-img-tick: $\langle \forall k. (\nexists k_0. f k_0 = k) \longrightarrow (\forall c. \neg(\text{hamlet } ((\text{Rep-run } r) \text{ } k \text{ } c))) \rangle$

and no-img-time: $\langle \bigwedge n. (\nexists n_0. f n_0 = (\text{Suc } n)) \longrightarrow (\forall c. \text{time } ((\text{Rep-run } r) \text{ } (\text{Suc } n) \text{ } c) = \text{time } ((\text{Rep-run } r) \text{ } n \text{ } c)) \rangle$

$n \text{ } c)) \rangle$

and fxge: $\langle \forall x. f x \geq x \rangle$ and f0n: $\langle \bigwedge n. f 0 \leq n \rangle$ and f0: $\langle f 0 = 0 \rangle$

by (auto simp add: dilating-def dilating-fun-def)

have finite-prefix: $\langle \bigwedge n. \text{finite } \{i. f i \leq n\} \rangle$ by (auto simp add: finite-less-ub fxge)

have prefix-not-empty: $\langle \bigwedge n. \{i. f i \leq n\} \neq \{\} \rangle$ using f0n by blast

have 1: $\langle \text{mono } (\text{dil-inverse } f) \rangle$

proof –

{ fix x::nat and y::nat assume hyp: $\langle x \leq y \rangle$

hence inc: $\langle \{i. f i \leq x\} \subseteq \{i. f i \leq y\} \rangle$

by (simp add: hyp Collect-mono le-trans)

from Max-mono[OF inc prefix-not-empty finite-prefix]

have $\langle (\text{dil-inverse } f) \text{ } x \leq (\text{dil-inverse } f) \text{ } y \rangle$ unfolding dil-inverse-def .

} thus ?thesis unfolding mono-def by simp

qed

from first-dilated-instant[OF smf f0] have 2: $\langle (\text{dil-inverse } f) \text{ } 0 = 0 \rangle$

unfolding dil-inverse-def .

from fxge have $\langle \forall n i. f i \leq n \longrightarrow i \leq n \rangle$ using le-trans by blast

hence 3: $\langle \forall n. (dil-inverse\ f)\ n \leq n \rangle$ **using** *Max-in[OF finite-prefix prefix-not-empty]*
unfolding *dil-inverse-def* **by** *blast*
 from 1 2 3 **have** *: $\langle contracting_fun\ (dil-inverse\ f) \rangle$ **by** (*simp add: contracting-fun-def*)
 have 4: $\langle \forall n\ c\ k. f\ ((dil-inverse\ f)\ n) < k \wedge k \leq n$
 $\longrightarrow \neg hamlet\ ((Rep-run\ r)\ k\ c) \rangle$
using *not-image-stut[OF assms]* *no-img-tick* **unfolding** *dil-inverse-def* **by** *blast*
 have 5: $\langle (\forall n\ c\ k. f\ ((dil-inverse\ f)\ n) \leq k \wedge k \leq n$
 $\longrightarrow time\ ((Rep-run\ r)\ k\ c) = time\ ((Rep-run\ sub)\ ((dil-inverse$
 $f)\ n)\ c) \rangle$
proof –
 { **fix** $n\ c\ k$ **assume** $h: \langle f\ ((dil-inverse\ f)\ n) \leq k \wedge k \leq n$
 $let\ ?\tau = \langle time\ (Rep-run\ sub)\ ((dil-inverse\ f)\ n)\ c \rangle$
have $tau: \langle time\ (Rep-run\ sub)\ ((dil-inverse\ f)\ n)\ c \rangle = ?\tau \rangle$..
have $gn: \langle (dil-inverse\ f)\ n = Max\ \{i. f\ i \leq n\} \rangle$ **unfolding** *dil-inverse-def* ..
from *time-stuttering[OF assms tau, of k]* *not-image-stut[OF assms gn]*
have $\langle time\ ((Rep-run\ r)\ k\ c) = time\ ((Rep-run\ sub)\ ((dil-inverse\ f)\ n)\ c) \rangle$
proof (*cases* $\langle f\ ((dil-inverse\ f)\ n) = k \rangle$)
 case *True*
 moreover **have** $\langle \forall n\ c. time\ (Rep-run\ sub\ n\ c) = time\ (Rep-run\ r\ (f\ n)$
 $c) \rangle$
 using *assms* **by** (*simp add: dilating-def*)
 ultimately show *?thesis* **by** *simp*
 next
 case *False*
 with h **have** $\langle f\ (Max\ \{i. f\ i \leq n\}) < k \wedge k \leq n \rangle$ **by** (*simp add:*
 dil-inverse-def)
 with *time-stuttering[OF assms tau, of k]* *not-image-stut[OF assms gn]*
 show *?thesis* **unfolding** *dil-inverse-def* **by** *auto*
 qed
 } **thus** *?thesis* **by** *simp*
 qed
 from * 5 4 **show** *?thesis* **unfolding** *contracting-def* **by** *simp*
qed
end

7.1.4 Main Theorems

theory *Stuttering*
imports *StutteringLemmas*

begin

Sporadic specifications are preserved in a dilated run.

lemma *sporadic-sub*:

assumes $\langle sub \ll r \rangle$

and $\langle sub \in \llbracket c \text{ sporadic } \tau \text{ on } c \rrbracket_{TESL} \rangle$

shows $\langle r \in \llbracket c \text{ sporadic } \tau \text{ on } c \rrbracket_{TESL} \rangle$

proof –

from *assms(1)* *is-subrun-def* **obtain** f

where $\langle dilating f sub r \rangle$ **by** *blast*

hence $\langle \forall n. c. time ((Rep-run sub) n c) = time ((Rep-run r) (f n) c) \rangle$

$\wedge hamlet ((Rep-run sub) n c) = hamlet ((Rep-run r) (f n) c) \rangle$ **by** (*simp*

add: dilating-def)

moreover from *assms(2)* **have**

$\langle sub \in \{r. \exists n. hamlet ((Rep-run r) n c) \wedge time ((Rep-run r) n c') = \tau\} \rangle$ **by**

simp

from this obtain k **where** $\langle time ((Rep-run sub) k c') = \tau \wedge hamlet ((Rep-run sub) k c) \rangle$ **by** *auto*

ultimately have $\langle time ((Rep-run r) (f k) c') = \tau \wedge hamlet ((Rep-run r) (f k) c) \rangle$ **by** *simp*

thus *?thesis* **by** *auto*

qed

Implications are preserved in a dilated run.

theorem *implies-sub*:

assumes $\langle sub \ll r \rangle$

and $\langle sub \in \llbracket c_1 \text{ implies } c_2 \rrbracket_{TESL} \rangle$

shows $\langle r \in \llbracket c_1 \text{ implies } c_2 \rrbracket_{TESL} \rangle$

proof –

from *assms(1)* *is-subrun-def* **obtain** f **where** $\langle dilating f sub r \rangle$ **by** *blast*

moreover from *assms(2)* **have**

$\langle sub \in \{r. \forall n. hamlet ((Rep-run r) n c_1) \longrightarrow hamlet ((Rep-run r) n c_2)\} \rangle$ **by**

simp

hence $\langle \forall n. hamlet ((Rep-run sub) n c_1) \longrightarrow hamlet ((Rep-run sub) n c_2) \rangle$ **by**

simp

ultimately have $\langle \forall n. hamlet ((Rep-run r) n c_1) \longrightarrow hamlet ((Rep-run r) n c_2) \rangle$

using *ticks-imp-ticks-subk ticks-sub* **by** *blast*

thus *?thesis* **by** *simp*

qed

theorem *implies-not-sub*:

assumes $\langle sub \ll r \rangle$

and $\langle sub \in \llbracket c_1 \text{ implies not } c_2 \rrbracket_{TESL} \rangle$

shows $\langle r \in \llbracket c_1 \text{ implies not } c_2 \rrbracket_{TESL} \rangle$

proof –

from *assms(1)* *is-subrun-def* **obtain** f **where** $\langle dilating f sub r \rangle$ **by** *blast*

moreover from *assms(2)* **have**

$\langle sub \in \{r. \forall n. hamlet ((Rep-run r) n c_1) \longrightarrow \neg hamlet ((Rep-run r) n c_2)\} \rangle$

by *simp*

hence $\langle \forall n. hamlet ((Rep-run sub) n c_1) \longrightarrow \neg hamlet ((Rep-run sub) n c_2) \rangle$ **by**

simp

ultimately have $\langle \forall n. \text{hamlet } ((\text{Rep-run } r) \ n \ c_1) \longrightarrow \neg \text{hamlet } ((\text{Rep-run } r) \ n \ c_2) \rangle$
 using *ticks-imp-ticks-subk ticks-sub* by *blast*
 thus *?thesis* by *simp*
 qed

Precedence relations are preserved in a dilated run.

theorem *weakly-precedes-sub*:

assumes $\langle \text{sub} \ll r \rangle$
 and $\langle \text{sub} \in \llbracket c_1 \text{ weakly precedes } c_2 \rrbracket_{TESL} \rangle$
 shows $\langle r \in \llbracket c_1 \text{ weakly precedes } c_2 \rrbracket_{TESL} \rangle$

proof –

from *assms(1) is-subrun-def* obtain *f* where $\ast: \langle \text{dilating } f \text{ sub } r \rangle$ by *blast*

from *assms(2)* have

$\langle \text{sub} \in \{ r. \forall n. (\text{run-tick-count } r \ c_2 \ n) \leq (\text{run-tick-count } r \ c_1 \ n) \} \rangle$ by *simp*

hence $\langle \forall n. (\text{run-tick-count } \text{sub} \ c_2 \ n) \leq (\text{run-tick-count } \text{sub} \ c_1 \ n) \rangle$ by *simp*

from *dil-tick-count[OF assms(1) this]* have $\langle \forall n. (\text{run-tick-count } r \ c_2 \ n) \leq (\text{run-tick-count } r \ c_1 \ n) \rangle$ by *simp*

thus *?thesis* by *simp*

qed

theorem *strictly-precedes-sub*:

assumes $\langle \text{sub} \ll r \rangle$
 and $\langle \text{sub} \in \llbracket c_1 \text{ strictly precedes } c_2 \rrbracket_{TESL} \rangle$
 shows $\langle r \in \llbracket c_1 \text{ strictly precedes } c_2 \rrbracket_{TESL} \rangle$

proof –

from *assms(1) is-subrun-def* obtain *f* where $\ast: \langle \text{dilating } f \text{ sub } r \rangle$ by *blast*

from *assms(2)* have $\langle \text{sub} \in \{ \varrho. \forall n::\text{nat}. (\text{run-tick-count } \varrho \ c_2 \ n) \leq (\text{run-tick-count-strictly } \varrho \ c_1 \ n) \} \rangle$ by *simp*

with *strictly-precedes-alt-def2[of $\langle c_2 \rangle \langle c_1 \rangle$]* have

$\langle \text{sub} \in \{ \varrho. (\neg \text{hamlet } ((\text{Rep-run } \varrho) \ 0 \ c_2)) \wedge (\forall n::\text{nat}. (\text{run-tick-count } \varrho \ c_2 \ (\text{Suc } n)) \leq (\text{run-tick-count } \varrho \ c_1 \ n)) \} \rangle$

by *blast*

hence $\langle (\neg \text{hamlet } ((\text{Rep-run } \text{sub}) \ 0 \ c_2)) \wedge (\forall n::\text{nat}. (\text{run-tick-count } \text{sub} \ c_2 \ (\text{Suc } n)) \leq (\text{run-tick-count } \text{sub} \ c_1 \ n)) \rangle$

by *simp*

hence

$1: \langle (\neg \text{hamlet } ((\text{Rep-run } \text{sub}) \ 0 \ c_2)) \wedge (\forall n::\text{nat}. (\text{tick-count } \text{sub} \ c_2 \ (\text{Suc } n)) \leq (\text{tick-count } \text{sub} \ c_1 \ n)) \rangle$

by (*simp add: tick-count-is-fun*)

have $\langle \forall n::\text{nat}. (\text{tick-count } r \ c_2 \ (\text{Suc } n)) \leq (\text{tick-count } r \ c_1 \ n) \rangle$

proof –

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

have $\langle \text{tick-count } r \ c_2 \ (\text{Suc } n) \leq \text{tick-count } r \ c_1 \ n \rangle$

proof (*cases $\langle \exists n_0. f \ n_0 = n \rangle$*)

case *True* — *n* is in the image of *f*

from *this* obtain n_0 where $fn: \langle f \ n_0 = n \rangle$ by *blast*

show *?thesis*

proof (*cases $\langle \exists sn_0. f \ sn_0 = \text{Suc } n \rangle$*)

case *True* — *Suc n* is in the image of *f*
 from *this* obtain *sn₀* where *fsn*: $\langle f\ sn_0 = \text{Suc } n \rangle$ by *blast*
 with *fn* have $\langle sn_0 = \text{Suc } n_0 \rangle$ using *strict-mono-suc* * *dilating-def*
dilating-fun-def by *blast*
 with 1 have $\langle \text{tick-count sub } c_2\ sn_0 \leq \text{tick-count sub } c_1\ n_0 \rangle$ by *simp*
 thus ?thesis using *fn fsn tick-count-sub*[*OF* *] by *simp*
 next
 case *False* — *Suc n* is not in the image of *f*
 hence $\langle \neg \text{hamlet } ((\text{Rep-run } r) (\text{Suc } n) c_2) \rangle$
 using * by (*simp add: dilating-def dilating-fun-def*)
 hence $\langle \text{tick-count } r\ c_2\ (\text{Suc } n) = \text{tick-count } r\ c_2\ n \rangle$ by (*simp add:*
tick-count-suc)
 also have $\langle \dots = \text{tick-count sub } c_2\ n_0 \rangle$ using *fn tick-count-sub*[*OF* *]
 by *simp*
 finally have $\langle \text{tick-count } r\ c_2\ (\text{Suc } n) = \text{tick-count sub } c_2\ n_0 \rangle$.
 moreover have $\langle \text{tick-count sub } c_2\ n_0 \leq \text{tick-count sub } c_2\ (\text{Suc } n_0) \rangle$
 by (*simp add: tick-count-suc*)
 ultimately have $\langle \text{tick-count } r\ c_2\ (\text{Suc } n) \leq \text{tick-count sub } c_2\ (\text{Suc } n_0) \rangle$
 by *simp*
 moreover have $\langle \text{tick-count sub } c_2\ (\text{Suc } n_0) \leq \text{tick-count sub } c_1\ n_0 \rangle$
 using 1 by *simp*
 ultimately have $\langle \text{tick-count } r\ c_2\ (\text{Suc } n) \leq \text{tick-count sub } c_1\ n_0 \rangle$ by
simp
 thus ?thesis using *tick-count-sub*[*OF* *] *fn* by *simp*
 qed
 next
 case *False* — *n* is not in the image of *f*
 from *greatest-prev-image*[*OF* * *this*] obtain *np*
 where *np-prop*: $\langle f\ n_p < n \wedge (\forall k. f\ n_p < k \wedge k \leq n \longrightarrow (\nexists k_0. f\ k_0 = k)) \rangle$ by *blast*
 from *tick-count-latest*[*OF* * *this*] have $\langle \text{tick-count } r\ c_1\ n = \text{tick-count } r\ c_1\ (f\ n_p) \rangle$.
 hence *a*: $\langle \text{tick-count } r\ c_1\ n = \text{tick-count sub } c_1\ n_p \rangle$ using *tick-count-sub*[*OF*
 *] by *simp*
 have *b*: $\langle \text{tick-count sub } c_2\ (\text{Suc } n_p) \leq \text{tick-count sub } c_1\ n_p \rangle$ using 1 by
simp
 show ?thesis
 proof (cases $\langle \exists sn_0. f\ sn_0 = \text{Suc } n \rangle$)
 case *True* — *Suc n* is in the image of *f*
 from *this* obtain *sn₀* where *fsn*: $\langle f\ sn_0 = \text{Suc } n \rangle$ by *blast*
 from *next-non-stuttering*[*OF* * *np-prop this*] have *sn-prop*: $\langle sn_0 = \text{Suc } n_p \rangle$.
 with *b* have $\langle \text{tick-count sub } c_2\ sn_0 \leq \text{tick-count sub } c_1\ n_p \rangle$ by *simp*
 thus ?thesis using *tick-count-sub*[*OF* *] *fsn a* by *auto*
 next
 case *False* — *Suc n* is not in the image of *f*
 hence $\langle \neg \text{hamlet } ((\text{Rep-run } r) (\text{Suc } n) c_2) \rangle$
 using * by (*simp add: dilating-def dilating-fun-def*)
 hence $\langle \text{tick-count } r\ c_2\ (\text{Suc } n) = \text{tick-count } r\ c_2\ n \rangle$ by (*simp add:*

tick-count-suc)
also have $\langle \dots = \text{tick-count sub } c_2 \ n_p \rangle$ **using** *np-prop tick-count-sub*[*OF*
 $*$]
by (*simp add: tick-count-latest*[*OF* $*$ *np-prop*])
finally have $\langle \text{tick-count } r \ c_2 \ (\text{Suc } n) = \text{tick-count sub } c_2 \ n_p \rangle$.
moreover have $\langle \text{tick-count sub } c_2 \ n_p \leq \text{tick-count sub } c_2 \ (\text{Suc } n_p) \rangle$
by (*simp add: tick-count-suc*)
ultimately have $\langle \text{tick-count } r \ c_2 \ (\text{Suc } n) \leq \text{tick-count sub } c_2 \ (\text{Suc } n_p) \rangle$
by simp
moreover have $\langle \text{tick-count sub } c_2 \ (\text{Suc } n_p) \leq \text{tick-count sub } c_1 \ n_p \rangle$
using 1 by simp
ultimately have $\langle \text{tick-count } r \ c_2 \ (\text{Suc } n) \leq \text{tick-count sub } c_1 \ n_p \rangle$ **by**
simp
thus ?thesis using np-prop mono-tick-count using a by linarith
qed
qed
} thus ?thesis ..
qed
moreover from 1 have $\langle \neg \text{hamlet } ((\text{Rep-run } r) \ 0 \ c_2) \rangle$
using * empty-dilated-prefix ticks-sub by fastforce
ultimately show ?thesis by (simp add: tick-count-is-fun strictly-precedes-alt-def2)
qed

Time delayed relations are preserved in a dilated run.

theorem *time-delayed-sub*:

assumes $\langle \text{sub} \ll r \rangle$
and $\langle \text{sub} \in \llbracket a \text{ time-delayed by } \delta\tau \text{ on ms implies } b \rrbracket_{\text{TESL}} \rangle$
shows $\langle r \in \llbracket a \text{ time-delayed by } \delta\tau \text{ on ms implies } b \rrbracket_{\text{TESL}} \rangle$
proof –
from *assms(1) is-subrun-def* **obtain** *f* **where** $\ast : \langle \text{dilating } f \text{ sub } r \rangle$ **by blast**
from *assms(2)* **have** $\langle \forall n. \text{hamlet } ((\text{Rep-run sub}) \ n \ a) \rangle$
 $\longrightarrow (\forall m \geq n. \text{first-time sub ms } m \ (\text{time } ((\text{Rep-run sub}) \ n \ ms) + \delta\tau) \longrightarrow \text{hamlet } ((\text{Rep-run sub}) \ m \ b)) \rangle$
using *TESL-interpretation-atomic.simps(5)*[*of* $\langle a \rangle \langle \delta\tau \rangle \langle ms \rangle \langle b \rangle$] **by simp**
hence $\ast : \langle \forall n_0. \text{hamlet } ((\text{Rep-run } r) \ (f \ n_0) \ a) \rangle$
 $\longrightarrow (\forall m_0 \geq n_0. \text{first-time } r \ ms \ (f \ m_0) \ (\text{time } ((\text{Rep-run } r) \ (f \ n_0) \ ms) + \delta\tau) \longrightarrow \text{hamlet } ((\text{Rep-run } r) \ (f \ m_0) \ b)) \rangle$
using *first-time-image*[*OF* $*$] *dilating-def* $*$ **by fastforce**
hence $\langle \forall n. \text{hamlet } ((\text{Rep-run } r) \ n \ a) \rangle$
 $\longrightarrow (\forall m \geq n. \text{first-time } r \ ms \ m \ (\text{time } ((\text{Rep-run } r) \ n \ ms) + \delta\tau) \longrightarrow \text{hamlet } ((\text{Rep-run } r) \ m \ b)) \rangle$
proof –
{ fix *n* **assume** *assm*: $\langle \text{hamlet } ((\text{Rep-run } r) \ n \ a) \rangle$
from *ticks-image-sub*[*OF* $*$ *assm*] **obtain** *n₀* **where** $nfn0 : \langle n = f \ n_0 \rangle$ **by blast**
with \ast *assm* **have** *ft0*:
 $\langle (\forall m_0 \geq n_0. \text{first-time } r \ ms \ (f \ m_0) \ (\text{time } ((\text{Rep-run } r) \ (f \ n_0) \ ms) + \delta\tau) \longrightarrow \text{hamlet } ((\text{Rep-run } r) \ m \ b)) \rangle$

```

       $\longrightarrow$  hamlet  $((\text{Rep-run } r) (f m_0) b))$  by blast
have  $\langle (\forall m \geq n. \text{first-time } r \text{ ms } m (\text{time } ((\text{Rep-run } r) n \text{ ms}) + \delta\tau) \longrightarrow \text{hamlet } ((\text{Rep-run } r) m b)) \rangle$ 
proof -
{ fix m assume hyp:  $\langle m \geq n \rangle$ 
  have  $\langle \text{first-time } r \text{ ms } m (\text{time } (\text{Rep-run } r n \text{ ms}) + \delta\tau) \longrightarrow \text{hamlet } (\text{Rep-run } r m b) \rangle$ 
  proof (cases  $\langle \exists m_0. f m_0 = m \rangle$ )
  case True
    from this obtain m0 where  $\langle m = f m_0 \rangle$  by blast
    moreover have  $\langle \text{strict-mono } f \rangle$  using * by (simp add: dilating-def dilating-fun-def)
    ultimately show ?thesis using ft0 hyp nfn0 by (simp add: strict-mono-less-eq)
  next
  case False thus ?thesis
  proof (cases  $\langle m = 0 \rangle$ )
  case True
    hence  $\langle m = f 0 \rangle$  using * by (simp add: dilating-def dilating-fun-def)
    then show ?thesis using False by blast
  next
  case False
    hence  $\langle \exists pm. m = \text{Suc } pm \rangle$  by (simp add: not0-implies-Suc)
    from this obtain pm where  $\langle m = \text{Suc } pm \rangle$  by blast
    hence  $\langle \exists pm_0. f pm_0 = \text{Suc } pm \rangle$  using  $\langle \exists m_0. f m_0 = m \rangle$  by simp
    with * have  $\langle \text{time } (\text{Rep-run } r (\text{Suc } pm) \text{ ms}) = \text{time } (\text{Rep-run } r pm \text{ ms}) \rangle$ 
    using dilating-def dilating-fun-def by blast
    hence  $\langle \text{time } (\text{Rep-run } r pm \text{ ms}) = \text{time } (\text{Rep-run } r m \text{ ms}) \rangle$  using mpm
  by simp
    moreover from mpm have  $\langle pm < m \rangle$  by simp
    ultimately have  $\langle \exists m' < m. \text{time } (\text{Rep-run } r m' \text{ ms}) = \text{time } (\text{Rep-run } r m \text{ ms}) \rangle$  by blast
    hence  $\langle \neg(\text{first-time } r \text{ ms } m (\text{time } (\text{Rep-run } r n \text{ ms}) + \delta\tau)) \rangle$ 
    by (auto simp add: first-time-def)
    thus ?thesis by simp
  qed
qed
} thus ?thesis by simp
qed
} thus ?thesis by simp
qed
thus ?thesis by simp
qed

```

Time relations are preserved by contraction

lemma *tagrel-sub-inv*:

```

assumes  $\langle \text{sub} \ll r \rangle$ 
and  $\langle r \in \llbracket \text{time-relation } [c_1, c_2] \in R \rrbracket_{TESL} \rangle$ 
shows  $\langle \text{sub} \in \llbracket \text{time-relation } [c_1, c_2] \in R \rrbracket_{TESL} \rangle$ 

```

proof –

from *assms(1) is-subrun-def* **obtain** *f* **where** $df: \langle \text{dilating } f \text{ sub } r \rangle$ **by** *blast*
moreover from *assms(2) TESL-interpretation-atomic.simps(2)* **have**
 $\langle r \in \{ \varrho. \forall n. R (\text{time } ((\text{Rep-run } \varrho) \ n \ c_1), \text{time } ((\text{Rep-run } \varrho) \ n \ c_2)) \} \rangle$ **by** *blast*
hence $\langle \forall n. R (\text{time } ((\text{Rep-run } r) \ n \ c_1), \text{time } ((\text{Rep-run } r) \ n \ c_2)) \rangle$ **by** *simp*
hence $\langle \forall n. (\exists n_0. f \ n_0 = n) \longrightarrow R (\text{time } ((\text{Rep-run } r) \ n \ c_1), \text{time } ((\text{Rep-run } r) \ n \ c_2)) \rangle$ **by** *simp*
hence $\langle \forall n_0. R (\text{time } ((\text{Rep-run } r) \ (f \ n_0) \ c_1), \text{time } ((\text{Rep-run } r) \ (f \ n_0) \ c_2)) \rangle$ **by** *blast*
moreover from *dilating-def df* **have**
 $\langle \forall n \ c. \text{time } ((\text{Rep-run } \text{sub}) \ n \ c) = \text{time } ((\text{Rep-run } r) \ (f \ n) \ c) \rangle$ **by** *blast*
ultimately have $\langle \forall n_0. R (\text{time } ((\text{Rep-run } \text{sub}) \ n_0 \ c_1), \text{time } ((\text{Rep-run } \text{sub}) \ n_0 \ c_2)) \rangle$ **by** *auto*
thus *?thesis* **by** *simp*
qed

A time relation is preserved through dilation of a run.

lemma *tagrel-sub'*:

assumes $\langle \text{sub} \ll r \rangle$
and $\langle \text{sub} \in \llbracket \text{time-relation } [c_1, c_2] \in R \rrbracket_{\text{TESL}} \rangle$
shows $\langle R (\text{time } ((\text{Rep-run } r) \ n \ c_1), \text{time } ((\text{Rep-run } r) \ n \ c_2)) \rangle$

proof –

from *assms(1) is-subrun-def* **obtain** *f* **where** $\ast: \langle \text{dilating } f \text{ sub } r \rangle$ **by** *blast*
moreover from *assms(2) TESL-interpretation-atomic.simps(2)* **have**
 $\langle \text{sub} \in \{ r. \forall n. R (\text{time } ((\text{Rep-run } r) \ n \ c_1), \text{time } ((\text{Rep-run } r) \ n \ c_2)) \} \rangle$ **by** *blast*
hence $1: \langle \forall n. R (\text{time } ((\text{Rep-run } \text{sub}) \ n \ c_1), \text{time } ((\text{Rep-run } \text{sub}) \ n \ c_2)) \rangle$ **by** *simp*
show *?thesis*
proof (*induction n*)
case *0*
from *1* **have** $\langle R (\text{time } ((\text{Rep-run } \text{sub}) \ 0 \ c_1), \text{time } ((\text{Rep-run } \text{sub}) \ 0 \ c_2)) \rangle$ **by** *simp*
moreover from \ast **have** $\langle f \ 0 = 0 \rangle$ **by** (*simp add: dilating-def dilating-fun-def*)
moreover from \ast **have** $\langle \forall c. \text{time } ((\text{Rep-run } \text{sub}) \ 0 \ c) = \text{time } ((\text{Rep-run } r) \ (f \ 0) \ c) \rangle$
by (*simp add: dilating-def*)
ultimately show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
proof (*cases* $\langle \nexists n_0. f \ n_0 = \text{Suc } n \rangle$)
case *True*
with \ast **have** $\langle \forall c. \text{time } (\text{Rep-run } r \ (\text{Suc } n) \ c) = \text{time } (\text{Rep-run } r \ n \ c) \rangle$
by (*simp add: dilating-def dilating-fun-def*)
thus *?thesis* **using** *Suc.IH* **by** *simp*
next
case *False*
from *this* **obtain** n_0 **where** $n_0 \text{prop}: \langle f \ n_0 = \text{Suc } n \rangle$ **by** *blast*
from *1* **have** $\langle R (\text{time } ((\text{Rep-run } \text{sub}) \ n_0 \ c_1), \text{time } ((\text{Rep-run } \text{sub}) \ n_0 \ c_2)) \rangle$
by *simp*

moreover from $n_0 \text{prop} *$ have $\langle \text{time} ((\text{Rep-run sub}) n_0 c_1) = \text{time} ((\text{Rep-run } r) (\text{Suc } n) c_1) \rangle$
 by (simp add: dilating-def)
 moreover from $n_0 \text{prop} *$ have $\langle \text{time} ((\text{Rep-run sub}) n_0 c_2) = \text{time} ((\text{Rep-run } r) (\text{Suc } n) c_2) \rangle$
 by (simp add: dilating-def)
 ultimately show ?thesis by simp
 qed
 qed
 qed

corollary *tagrel-sub*:

assumes $\langle \text{sub} \ll r \rangle$
 and $\langle \text{sub} \in \llbracket \text{time-relation } [c_1, c_2] \in R \rrbracket_{\text{TESL}} \rangle$
 shows $\langle r \in \llbracket \text{time-relation } [c_1, c_2] \in R \rrbracket_{\text{TESL}} \rangle$
 using *tagrel-sub*'[OF *assms*] unfolding *TESL-interpretation-atomic.simps*(3) by simp

theorem *kill-sub*:

assumes $\langle \text{sub} \ll r \rangle$
 and $\langle \text{sub} \in \llbracket c_1 \text{ kills } c_2 \rrbracket_{\text{TESL}} \rangle$
 shows $\langle r \in \llbracket c_1 \text{ kills } c_2 \rrbracket_{\text{TESL}} \rangle$
proof –
 from *assms*(1) *is-subrun-def* obtain *f* where $\langle \text{dilating } f \text{ sub } r \rangle$ by blast
 from *assms*(2) *TESL-interpretation-atomic.simps*(8) have
 $\langle \forall n. \text{hamlet } (\text{Rep-run sub } n c_1) \longrightarrow (\forall m \geq n. \neg \text{hamlet } (\text{Rep-run sub } m c_2)) \rangle$
 by simp
 hence 1: $\langle \forall n. \text{hamlet } (\text{Rep-run } r (f n) c_1) \longrightarrow (\forall m \geq n. \neg \text{hamlet } (\text{Rep-run } r (f m) c_2)) \rangle$
 using *ticks-sub*[OF *] by simp
 hence $\langle \forall n. \text{hamlet } (\text{Rep-run } r (f n) c_1) \longrightarrow (\forall m \geq (f n). \neg \text{hamlet } (\text{Rep-run } r m c_2)) \rangle$
proof –
 { fix *n* assume $\langle \text{hamlet } (\text{Rep-run } r (f n) c_1) \rangle$
 with 1 have 2: $\langle \forall m \geq n. \neg \text{hamlet } (\text{Rep-run } r (f m) c_2) \rangle$ by simp
 have $\langle \forall m \geq (f n). \neg \text{hamlet } (\text{Rep-run } r m c_2) \rangle$
proof –
 { fix *m* assume $h: \langle m \geq f n \rangle$
 have $\langle \neg \text{hamlet } (\text{Rep-run } r m c_2) \rangle$
proof (cases $\langle \exists m_0. f m_0 = m \rangle$)
 case True
 from this obtain m_0 where $f m_0: \langle f m_0 = m \rangle$ by blast
 hence $\langle m_0 \geq n \rangle$
 using * *dilating-def* *dilating-fun-def* *h* *strict-mono-less-eq* by fastforce
 with 2 show ?thesis using *f m_0* by blast
 next
 case False
 thus ?thesis using *ticks-image-sub*'[OF *] by blast
 qed
 }
 qed

```

    } thus ?thesis by simp
  qed
} thus ?thesis by simp
qed
hence  $\langle \forall n. \text{hamlet} (\text{Rep-run } r \ n \ c_1) \longrightarrow (\forall m \geq n. \neg \text{hamlet} (\text{Rep-run } r \ m \ c_2)) \rangle$ 
  using ticks-imp-ticks-subk[OF *] by blast
thus ?thesis using TESL-interpretation-atomic.simps(8) by blast
qed

```

```

lemma atomic-sub:
  assumes  $\langle \text{sub} \ll r \rangle$ 
    and  $\langle \text{sub} \in \llbracket \varphi \rrbracket_{\text{TESL}} \rangle$ 
    shows  $\langle r \in \llbracket \varphi \rrbracket_{\text{TESL}} \rangle$ 
proof (cases  $\varphi$ )
  case (SporadicOn)
    thus ?thesis using assms(2) sporadic-sub[OF assms(1)] by simp
  next
  case (TagRelation)
    thus ?thesis using assms(2) tagrel-sub[OF assms(1)] by simp
  next
  case (Implies)
    thus ?thesis using assms(2) implies-sub[OF assms(1)] by simp
  next
  case (ImpliesNot)
    thus ?thesis using assms(2) implies-not-sub[OF assms(1)] by simp
  next
  case (TimeDelayedBy)
    thus ?thesis using assms(2) time-delayed-sub[OF assms(1)] by simp
  next
  case (WeaklyPrecedes)
    thus ?thesis using assms(2) weakly-precedes-sub[OF assms(1)] by simp
  next
  case (StrictlyPrecedes)
    thus ?thesis using assms(2) strictly-precedes-sub[OF assms(1)] by simp
  next
  case (Kills)
    thus ?thesis using assms(2) kill-sub[OF assms(1)] by simp
qed

```

```

theorem TESL-stuttering-invariant:
  assumes  $\langle \text{sub} \ll r \rangle$ 
    shows  $\langle \text{sub} \in \llbracket S \rrbracket_{\text{TESL}} \implies r \in \llbracket S \rrbracket_{\text{TESL}} \rangle$ 
proof (induction S)
  case Nil
    thus ?case by simp
  next
  case (Cons a s)
    from Cons.premis have sa:  $\langle \text{sub} \in \llbracket a \rrbracket_{\text{TESL}} \rangle$  and sb:  $\langle \text{sub} \in \llbracket s \rrbracket_{\text{TESL}} \rangle$ 
    using TESL-interpretation-image by simp+

```

from *Cons.IH*[*OF sb*] **have** $\langle r \in \llbracket s \rrbracket_{TESL} \rangle$.
moreover from *atomic-sub*[*OF assms*(1) *sa*] **have** $\langle r \in \llbracket a \rrbracket_{TESL} \rangle$.
ultimately show *?case* **using** *TESL-interpretation-image* **by** *simp*
qed
end

Bibliography

- [1] F. Boulanger, C. Jacquet, C. Hardebolle, and I. Prodan. TESL: a language for reconciling heterogeneous execution traces. In *Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2014)*, pages 114–123, Lausanne, Switzerland, Oct 2014.
- [2] H. Nguyen Van, T. Balabonski, F. Boulanger, C. Keller, B. Valiron, and B. Wolff. A symbolic operational semantics for TESL with an application to heterogeneous system testing. In *Formal Modeling and Analysis of Timed Systems, 15th International Conference FORMATS 2017*, volume 10419 of *LNCS*. Springer, Sep 2017.