A Formal Development of a Polychronous Polytimed Coordination Language

Hai NGuyen Van Frederic Boulanger Burkhart Wolff

April 8, 2019

Contents

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 dataflow 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).

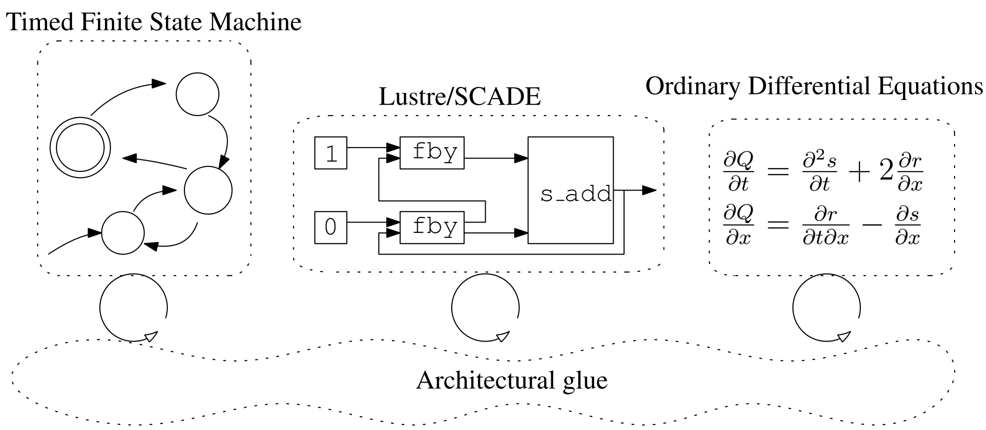

Figure 1.1: A Heterogeneous Timed System Model

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: $\Sigma^\infty = \mathbb{N} \to \mathcal{K} \to (\mathbb{B} \times \mathcal{T})$, 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 every time clock `c1` ticks, `c2` must have a tick at an 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 closk `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

[1] See 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.
- Instant indexes, (FIXME) which are natural integers, should not be used directly but appear here for technical and historical reasons.
- Tag constants are just constants of a type which denotes the metric time space.
- Tag variables represent the time at a given instant on a given clock.
- Tag expressions are used to represent either a tag constant or a delayed time with respect to a tag variable.

datatype *clock* = *Clk* ‹*string*›
type-synonym *instant-index* = ‹*nat*›

datatype $'\tau$ *tag-const* =

TConst $'\tau$ (τ_{cst})

datatype *tag-var* =
TSchematic ‹*clock* ∗ *instant-index*› (τ_{var})

2.1.2 Operators for the TESL language

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

datatype $'\tau$ *TESL-atomic* =
SporadicOn ‹*clock*› ‹$'\tau$ *tag-const*› ‹*clock*› (*- sporadic - on - 55*)
| *TagRelation* ‹*clock*› ‹*clock*› ‹($'\tau$ *tag-const* × $'\tau$ *tag-const*) ⇒ *bool*›
(*time−relation* ⌊-, -⌋ ∈ *- 55*)
| *Implies* ‹*clock*› ‹*clock*› (**infixr** *implies 55*)
| *ImpliesNot* ‹*clock*› ‹*clock*› (**infixr** *implies not 55*)
| *TimeDelayedBy* ‹*clock*› ‹$'\tau$ *tag-const*› ‹*clock*› ‹*clock*› (*- time−delayed by - on - implies - 55*)
| *WeaklyPrecedes* ‹*clock*› ‹*clock*› (**infixr** *weakly precedes 55*)
| *StrictlyPrecedes* ‹*clock*› ‹*clock*› (**infixr** *strictly precedes 55*)
| *Kills* ‹*clock*› ‹*clock*› (**infixr** *kills 55*)

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

type-synonym $'\tau$ *TESL-formula* = ‹$'\tau$ *TESL-atomic list*›

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

fun *positive-atom* :: ‹$'\tau$ *TESL-atomic* ⇒ *bool*› **where**
‹*positive-atom* (*- sporadic - on -*) = *True*›
| ‹*positive-atom -* = *False*›

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

abbreviation *NoSporadic* :: ‹$'\tau$ *TESL-formula* ⇒ $'\tau$ *TESL-formula*› **where**
‹*NoSporadic f* ≡ (*List.filter* (λf_{atom}. *case* f_{atom} *of*
- sporadic - on - ⇒ *False*
| *-* ⇒ *True*) *f*)›

2.1.3 Field Structure of the Metric Time Space

In order to handle tag relations and delays, tag must be in a field. We show here that this is the case when the type parameter of $'\tau$ *tag-const* is itself a field.

instantiation *tag-const* :: (*plus*)*plus*
begin
fun *plus-tag-const* :: ‹$'a$ *tag-const* ⇒ $'a$ *tag-const* ⇒ $'a$ *tag-const*›

```
  where
    TConst-plus: ‹(TConst n) + (TConst p) = (TConst (n + p))›

  instance by (rule Groups.class.Groups.plus.of-class.intro)
end

instantiation tag-const :: (minus)minus
begin
  fun minus-tag-const :: ‹'a tag-const ⇒ 'a tag-const ⇒ 'a tag-const›
  where
    TConst-minus: ‹(TConst n) − (TConst p) = (TConst (n − p))›

  instance by (rule Groups.class.Groups.minus.of-class.intro)
end

instantiation tag-const :: (times)times
begin
  fun times-tag-const :: ‹'a tag-const ⇒ 'a tag-const ⇒ 'a tag-const›
  where
    TConst-times: ‹(TConst n) ∗ (TConst p) = (TConst (n ∗ p))›

  instance by (rule Groups.class.Groups.times.of-class.intro)
end

instantiation tag-const :: (divide)divide
begin
  fun divide-tag-const :: ‹'a tag-const ⇒ 'a tag-const ⇒ 'a tag-const›
  where
    TConst-divide: ‹divide (TConst n) (TConst p) = (TConst (divide n p))›

  instance by (rule Rings.class.Rings.divide.of-class.intro)
end

instantiation tag-const :: (inverse)inverse
begin
  fun inverse-tag-const :: ‹'a tag-const ⇒ 'a tag-const›
  where
    TConst-inverse: ‹inverse (TConst n) = (TConst (inverse n))›

  instance by (rule Fields.class.Fields.inverse.of-class.intro)
end

instantiation tag-const :: (order)order
begin
  inductive less-eq-tag-const :: ‹'a tag-const ⇒ 'a tag-const ⇒ bool›
  where
   Int-less-eq[simp]:      ‹n ≤ m ⟹ (TConst n) ≤ (TConst m)›

  definition less-tag: ‹(x::'a tag-const) < y ⟷ (x ≤ y) ∧ (x ≠ y)›
```

instance proof
  show ‹$\bigwedge x\ y$ :: $'a$ *tag-const*. $(x < y) = (x \leq y \wedge \neg\ y \leq x)$›
    using *less-eq-tag-const.simps less-tag* **by** *auto*
next
  { fix x::‹$'a$ *tag-const*›
    from *tag-const.exhaust* **obtain** x_0::$'a$ **where** *xx0*:‹$x = TConst\ x_0$› **by** *blast*
    with *Int-less-eq* **have** ‹$x \leq x$› **by** *simp*
  } thus $\bigwedge x$::$'a$ *tag-const*. $x \leq x$ **.**
next
  show ‹$\bigwedge x\ y\ z$ :: $'a$ *tag-const*. $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$›
    using *less-eq-tag-const.simps* **by** *auto*
next
  show ‹$\bigwedge x\ y$ :: $'a$ *tag-const*. $x \leq y \Longrightarrow y \leq x \Longrightarrow x = y$›
    using *less-eq-tag-const.simps* **by** *auto*
qed

end

instantiation *tag-const* :: (*linorder*)*linorder*
begin
instance proof
  { fix x::‹$'a$ *tag-const*› **and** y::‹$'a$ *tag-const*›
    from *tag-const.exhaust* **obtain** x_0::$'a$ **where** ‹$x = TConst\ x_0$› **by** *blast*
    moreover from *tag-const.exhaust* **obtain** y_0::$'a$ **where** ‹$y = TConst\ y_0$› **by** *blast*
    ultimately have ‹$x \leq y \vee y \leq x$› **using** *less-eq-tag-const.simps* **by** *fastforce*
  }
  thus ‹$\bigwedge x\ y$. (x::$'a$ *tag-const*) $\leq y \vee y \leq x$› **.**
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** ‹*hamlet* ≡ *fst*›
abbreviation *time* **where** ‹*time* ≡ *snd*›

type-synonym $'\tau$ *instant* = ‹*clock* ⇒ (*bool* × $'\tau$ *tag-const*)›

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

typedef (**overloaded**) $'\tau$::*linordered-field run* =
‹{ ϱ::*nat* ⇒ $'\tau$ *instant*. ∀ *c*. *mono* (λ*n*. *time* (ϱ *n c*)) }›
proof
show ‹(λ- -. (*True*, τ_{cst} *0*)) ∈ {ϱ. ∀ *c*. *mono* (λ*n*. *time* (ϱ *n c*))}›
unfolding *mono-def* **by** *blast*
qed

lemma *Abs-run-inverse-rewrite*:
‹∀ *c*. *mono* (λ*n*. *time* (ϱ *n c*)) ⟹ *Rep-run* (*Abs-run* ϱ) = ϱ›
by (*simp add*: *Abs-run-inverse*)

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

fun *run-tick-count* :: ‹($'\tau$::*linordered-field*) *run* ⇒ *clock* ⇒ *nat* ⇒ *nat*› ($\#_{\leq}$ - - -)
where
‹($\#_{\leq}$ ϱ *K 0*) = (*if hamlet* ((*Rep-run* ϱ) *0 K*)
then 1
else 0)›
| ‹($\#_{\leq}$ ϱ *K* (*Suc n*)) = (*if hamlet* ((*Rep-run* ϱ) (*Suc n*) *K*)
then 1 + ($\#_{\leq}$ ϱ *K n*)
else ($\#_{\leq}$ ϱ *K n*))›

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

fun *run-tick-count-strictly* :: ‹($'\tau$::*linordered-field*) *run* ⇒ *clock* ⇒ *nat* ⇒ *nat*› ($\#_{<}$ - - -)
where
‹($\#_{<}$ ϱ *K 0*) = *0*›
| ‹($\#_{<}$ ϱ *K* (*Suc n*)) = $\#_{\leq}$ ϱ *K n*›

definition *first-time* :: ‹$'a$::*linordered-field run* ⇒ *clock* ⇒ *nat* ⇒ $'a$ *tag-const* ⇒ *bool*›
where
‹*first-time* ϱ *K n* τ ≡ (*time* ((*Rep-run* ϱ) *n K*) = τ) ∧ (∄ *n'*. *n'* < *n* ∧ *time* ((*Rep-run* ϱ) *n' K*) = τ)›

lemma *before-first-time*:
assumes ‹*first-time* ϱ *K n* τ›
and ‹*m* < *n*›
shows ‹*time* ((*Rep-run* ϱ) *m K*) < τ›
proof −
have ‹*mono* (λ*n*. *time* (*Rep-run* ϱ *n K*))› **using** *Rep-run* **by** *blast*

moreover from *assms(2)* **have** ‹$m \leq n$› **using** *less-imp-le* **by** *simp*
moreover have ‹*mono* (λn. *time* (*Rep-run* ϱ n K))› **using** *Rep-run* **by** *blast*
ultimately have ‹*time* ((*Rep-run* ϱ) m K) $\leq$ *time* ((*Rep-run* ϱ) n K)› **by** (*simp add*:*mono-def*)
moreover from *assms(1)* **have** ‹*time* ((*Rep-run* ϱ) n K) $= \tau$› **using** *first-time-def* **by** *blast*
moreover from *assms* **have** ‹*time* ((*Rep-run* ϱ) m K) $\neq \tau$› **using** *first-time-def* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed

lemma *alt-first-time-def*:
assumes ‹$\forall m < n$. *time* ((*Rep-run* ϱ) m K) $< \tau$›
and ‹*time* ((*Rep-run* ϱ) n K) $= \tau$›
shows ‹*first-time* ϱ K n τ›
proof −
from *assms(1)* **have** ‹$\forall m < n$. *time* ((*Rep-run* ϱ) m K) $\neq \tau$› **by** (*simp add*: *less-le*)
with *assms(2)* **show** *?thesis* **by** (*simp add*: *first-time-def*)
qed

end

Chapter 3

Denotational Semantics

theory *Denotational*
imports
  TESL
  Run

begin

3.1 Denotational interpretation for atomic TESL formulae

fun *TESL-interpretation-atomic*
  $:: \langle({}'\tau::linordered\text{-}field)\ TESL\text{-}atomic \Rightarrow {}'\tau\ run\ set\rangle\ (\llbracket\ \text{-}\ \rrbracket_{TESL})$ **where**
  $\langle\llbracket\ K_1\ sporadic\ \tau\ on\ K_2\ \rrbracket_{TESL} =$
    $\{\ \varrho.\ \exists\, n::nat.\ hamlet\ ((Rep\text{-}run\ \varrho)\ n\ K_1) \wedge time\ ((Rep\text{-}run\ \varrho)\ n\ K_2) = \tau\ \}\rangle$
  $|\ \langle\llbracket\ time{-}relation\ \lfloor K_1,\ K_2\rfloor \in R\ \rrbracket_{TESL} =$
    $\{\ \varrho.\ \forall\, n::nat.\ R\ (time\ ((Rep\text{-}run\ \varrho)\ n\ K_1),\ time\ ((Rep\text{-}run\ \varrho)\ n\ K_2))\ \}\rangle$
  $|\ \langle\llbracket\ master\ implies\ slave\ \rrbracket_{TESL} =$
    $\{\ \varrho.\ \forall\, n::nat.\ hamlet\ ((Rep\text{-}run\ \varrho)\ n\ master) \longrightarrow hamlet\ ((Rep\text{-}run\ \varrho)\ n$ $slave)\ \}\rangle$
  $|\ \langle\llbracket\ master\ implies\ not\ slave\ \rrbracket_{TESL} =$
    $\{\ \varrho.\ \forall\, n::nat.\ hamlet\ ((Rep\text{-}run\ \varrho)\ n\ master) \longrightarrow \neg\ hamlet\ ((Rep\text{-}run\ \varrho)\ n$ $slave)\ \}\rangle$
  $|\ \langle\llbracket\ master\ time{-}delayed\ by\ \delta\tau\ on\ measuring\ implies\ slave\ \rrbracket_{TESL} =$
  — When master ticks, let's call @termt_0 the current date on measuring. Then, at the first instant when the date on measuring is @termt_0+δt, slave has to tick.
    $\{\ \varrho.\ \forall\, n.\ hamlet\ ((Rep\text{-}run\ \varrho)\ n\ master) \longrightarrow$
      $(let\ measured\text{-}time = time\ ((Rep\text{-}run\ \varrho)\ n\ measuring)\ in$
      $\forall\, m \geq n.\ \ first\text{-}time\ \varrho\ measuring\ m\ (measured\text{-}time + \delta\tau)$
        $\longrightarrow hamlet\ ((Rep\text{-}run\ \varrho)\ m\ slave)$
      $)$
    $\}\rangle$
  $|\ \langle\llbracket\ K_1\ weakly\ precedes\ K_2\ \rrbracket_{TESL} =$

$\{\ \varrho.\ \forall\, n{::}nat.\ (run\text{-}tick\text{-}count\ \varrho\ K_2\ n) \leq (run\text{-}tick\text{-}count\ \varrho\ K_1\ n)\ \}\rangle$
$|\ \langle[\![\ K_1\ strictly\ precedes\ K_2\ ]\!]_{TESL} =$
$\{\ \varrho.\ \forall\, n{::}nat.\ (run\text{-}tick\text{-}count\ \varrho\ K_2\ n) \leq (run\text{-}tick\text{-}count\text{-}strictly\ \varrho\ K_1\ n)\ \}\rangle$
$|\ \langle[\![\ K_1\ kills\ K_2\ ]\!]_{TESL} =$
$\{\ \varrho.\ \forall\, n{::}nat.\ hamlet\ ((Rep\text{-}run\ \varrho)\ n\ K_1) \longrightarrow (\forall\, m{\geq}n.\ \neg\ hamlet\ ((Rep\text{-}run\ \varrho)\ m\ K_2))\ \}\rangle$

3.2 Denotational interpretation for TESL formulae

fun *TESL-interpretation* :: $\langle({}'\tau{::}linordered\text{-}field)\ TESL\text{-}formula \Rightarrow {}'\tau\ run\ set\rangle$ ($[\![[\ \text{-}\ ]\!]]_{TESL}$) **where**
$\langle[\![[\ []\ ]\!]]_{TESL} = \{\ \text{-}.\ True\ \}\rangle$
$|\ \langle[\![[\ \varphi\ \#\ \Phi\ ]\!]]_{TESL} = [\![\ \varphi\ ]\!]_{TESL} \cap [\![[\ \Phi\ ]\!]]_{TESL}\rangle$

lemma *TESL-interpretation-homo*:
$\langle[\![\ \varphi\ ]\!]_{TESL} \cap [\![[\ \Phi\ ]\!]]_{TESL} = [\![[\ \varphi\ \#\ \Phi\ ]\!]]_{TESL}\rangle$
by *auto*

3.2.1 Image interpretation lemma

theorem *TESL-interpretation-image*:
$\langle[\![[\ \Phi\ ]\!]]_{TESL} = \bigcap\ ((\lambda\varphi.\ [\![\ \varphi\ ]\!]_{TESL})\ `\ set\ \Phi)\rangle$
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*:
$\langle[\![[\ \Phi_1\ @\ \Phi_2\ ]\!]]_{TESL} = [\![[\ \Phi_1\ ]\!]]_{TESL} \cap [\![[\ \Phi_2\ ]\!]]_{TESL}\rangle$
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*:
$\langle [\![\, (\Phi_1 \ @ \ \Phi_2) \ @ \ \Phi_3 \,]\!]_{TESL} = [\![\, \Phi_1 \ @ \ (\Phi_2 \ @ \ \Phi_3) \,]\!]_{TESL} \rangle$
by *auto*

lemma *TESL-interp-commute*:
shows $\langle [\![\, \Phi_1 \ @ \ \Phi_2 \,]\!]_{TESL} = [\![\, \Phi_2 \ @ \ \Phi_1 \,]\!]_{TESL} \rangle$
by (*simp add*: *TESL-interp-homo-append inf-sup-aci*(*1*))

lemma *TESL-interp-left-commute*:
$\langle [\![\, \Phi_1 \ @ \ (\Phi_2 \ @ \ \Phi_3) \,]\!]_{TESL} = [\![\, \Phi_2 \ @ \ (\Phi_1 \ @ \ \Phi_3) \,]\!]_{TESL} \rangle$
unfolding *TESL-interp-homo-append* **by** *auto*

lemma *TESL-interp-idem*:
$\langle [\![\, \Phi \ @ \ \Phi \,]\!]_{TESL} = [\![\, \Phi \,]\!]_{TESL} \rangle$
using *TESL-interp-homo-append* **by** *auto*

lemma *TESL-interp-left-idem*:
$\langle [\![\, \Phi_1 \ @ \ (\Phi_1 \ @ \ \Phi_2) \,]\!]_{TESL} = [\![\, \Phi_1 \ @ \ \Phi_2 \,]\!]_{TESL} \rangle$
using *TESL-interp-homo-append* **by** *auto*

lemma *TESL-interp-right-idem*:
$\langle [\![\, (\Phi_1 \ @ \ \Phi_2) \ @ \ \Phi_2 \,]\!]_{TESL} = [\![\, \Phi_1 \ @ \ \Phi_2 \,]\!]_{TESL} \rangle$
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*:
$\langle [\![\, [] \ @ \ \Phi \,]\!]_{TESL} = [\![\, \Phi \,]\!]_{TESL} \rangle$
by *simp*

lemma *TESL-interp-neutral2*:
$\langle [\![\, \Phi \ @ \ [] \,]\!]_{TESL} = [\![\, \Phi \,]\!]_{TESL} \rangle$
by *simp*

3.4 Decreasing interpretation of TESL formulae

lemma *TESL-sem-decreases-head*:
$\langle [\![\, \Phi \,]\!]_{TESL} \supseteq [\![\, \varphi \ \# \ \Phi \,]\!]_{TESL} \rangle$
by *simp*

lemma *TESL-sem-decreases-tail*:
$\langle [\![\, \Phi \,]\!]_{TESL} \supseteq [\![\, \Phi \ @ \ [\varphi] \,]\!]_{TESL} \rangle$
by (*simp add*: *TESL-interp-homo-append*)

lemma ‹$\varphi \# \Phi = [\varphi] @ \Phi$› **by** *simp*

lemma *TESL-interp-formula-stuttering*:
  assumes ‹$\varphi \in$ *set* Φ›
    shows ‹$[\![[\; \varphi \;\#\; \Phi \;]\!]]_{TESL} = [\![[\; \Phi \;]\!]]_{TESL}$›
proof −
  have ‹$\varphi \;\#\; \Phi = [\varphi] \;@\; \Phi$› **by** *simp*
  hence ‹$[\![[\; \varphi \# \Phi \;]\!]]_{TESL} = [\![[\; [\varphi] \;]\!]]_{TESL} \cap [\![[\; \Phi \;]\!]]_{TESL}$› **using** *TESL-interp-homo-append*
by *simp*
  thus *?thesis* **using** *assms TESL-interpretation-image* **by** *fastforce*
qed

lemma *TESL-interp-decreases*:
  ‹$[\![[\; \Phi \;]\!]]_{TESL} \supseteq [\![[\; \varphi \;\#\; \Phi \;]\!]]_{TESL}$›
by (*rule TESL-sem-decreases-head*)

lemma *TESL-interp-remdups-absorb*:
  ‹$[\![[\; \Phi \;]\!]]_{TESL} = [\![[\;$ *remdups* $\Phi \;]\!]]_{TESL}$›
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 ‹*set* Φ = *set* Φ'›
    shows ‹$[\![[\; \Phi \;]\!]]_{TESL} = [\![[\; \Phi' \;]\!]]_{TESL}$›
proof −
  have ‹*set* (*remdups* Φ) = *set* (*remdups* Φ')›
    by (*simp add*: *assms*)
  moreover have *fxpnt*Φ: ‹$\bigcap\; ((\lambda\varphi.\; [\![\; \varphi \;]\!]_{TESL})$ ‘ *set* $\Phi) = [\![[\; \Phi \;]\!]]_{TESL}$›
    by (*simp add*: *TESL-interpretation-image*)
  moreover have *fxpnt*Φ': ‹$\bigcap\; ((\lambda\varphi.\; [\![\; \varphi \;]\!]_{TESL})$ ‘ *set* $\Phi') = [\![[\; \Phi' \;]\!]]_{TESL}$›
    by (*simp add*: *TESL-interpretation-image*)
  moreover have ‹$\bigcap\; ((\lambda\varphi.\; [\![\; \varphi \;]\!]_{TESL})$ ‘ *set* $\Phi) = \bigcap\; ((\lambda\varphi.\; [\![\; \varphi \;]\!]_{TESL})$ ‘ *set* $\Phi')$›
    by (*simp add*: *assms*)
  ultimately show *?thesis* **using** *TESL-interp-remdups-absorb* **by** *auto*
qed

theorem *TESL-interp-decreases-setinc*:
  assumes ‹*set* $\Phi \subseteq$ *set* Φ'›
    shows ‹$[\![[\; \Phi \;]\!]]_{TESL} \supseteq [\![[\; \Phi' \;]\!]]_{TESL}$›
proof −
  obtain Φ_r **where** *decompose*: ‹*set* ($\Phi \;@\; \Phi_r$) = *set* Φ'› **using** *assms* **by** *auto*
  have ‹*set* ($\Phi \;@\; \Phi_r$) = *set* Φ'› **using** *assms decompose* **by** *blast*

moreover have ‹$(set\ \Phi) \cup (set\ \Phi_r) = set\ \Phi'$› **using** *assms decompose* **by** *auto*
moreover have ‹$[\![\ \Phi'\ ]\!]_{TESL} = [\![\ \Phi\ @\ \Phi_r\ ]\!]_{TESL}$› **using** *TESL-interp-set-lifting decompose* **by** *blast*
moreover have ‹$[\![\ \Phi\ @\ \Phi_r\ ]\!]_{TESL} = [\![\ \Phi\ ]\!]_{TESL} \cap [\![\ \Phi_r\ ]\!]_{TESL}$› **by** (*simp add*: *TESL-interp-homo-append*)
moreover have ‹$[\![\ \Phi\ ]\!]_{TESL} \supseteq [\![\ \Phi\ ]\!]_{TESL} \cap [\![\ \Phi_r\ ]\!]_{TESL}$› **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *TESL-interp-decreases-add-head*:
assumes ‹$set\ \Phi \subseteq set\ \Phi'$›
shows ‹$[\![\ \varphi\ \#\ \Phi\ ]\!]_{TESL} \supseteq [\![\ \varphi\ \#\ \Phi'\ ]\!]_{TESL}$›
using *assms TESL-interp-decreases-setinc* **by** *auto*

lemma *TESL-interp-decreases-add-tail*:
assumes ‹$set\ \Phi \subseteq set\ \Phi'$›
shows ‹$[\![\ \Phi\ @\ [\varphi]\ ]\!]_{TESL} \supseteq [\![\ \Phi'\ @\ [\varphi]\ ]\!]_{TESL}$›
using *TESL-interp-decreases-setinc*[*OF assms*]
by (*simp add*: *TESL-interpretation-image dual-order.trans*)

lemma *TESL-interp-absorb1*:
assumes ‹$set\ \Phi_1 \subseteq set\ \Phi_2$›
shows ‹$[\![\ \Phi_1\ @\ \Phi_2\ ]\!]_{TESL} = [\![\ \Phi_2\ ]\!]_{TESL}$›
by (*simp add*: *Int-absorb1 TESL-interp-decreases-setinc TESL-interp-homo-append assms*)

lemma *TESL-interp-absorb2*:
assumes ‹$set\ \Phi_2 \subseteq set\ \Phi_1$›
shows ‹$[\![\ \Phi_1\ @\ \Phi_2\ ]\!]_{TESL} = [\![\ \Phi_1\ ]\!]_{TESL}$›
using *TESL-interp-absorb1 TESL-interp-commute assms* **by** *blast*

3.5 Some special cases

lemma *NoSporadic-stable* [*simp*]:
‹$[\![\ \Phi\ ]\!]_{TESL} \subseteq [\![\ NoSporadic\ \Phi\ ]\!]_{TESL}$›
proof −
from *filter-is-subset* **have** ‹$set\ (NoSporadic\ \Phi) \subseteq set\ \Phi$› **.**
from *TESL-interp-decreases-setinc*[*OF this*] **show** *?thesis* **.**
qed

lemma *NoSporadic-idem* [*simp*]:
‹$[\![\ \Phi\ ]\!]_{TESL} \cap [\![\ NoSporadic\ \Phi\ ]\!]_{TESL} = [\![\ \Phi\ ]\!]_{TESL}$›
using *NoSporadic-stable* **by** *blast*

lemma *NoSporadic-setinc*:
‹$set\ (NoSporadic\ \Phi) \subseteq set\ \Phi$›
by (*rule filter-is-subset*)

end

theory *SymbolicPrimitive*
imports *Run*

begin
datatype *cnt-expr* =
TickCountLess ‹*clock*› ‹*instant-index*› $(\#^{<})$
| *TickCountLeq* ‹*clock*› ‹*instant-index*› $(\#^{\leq})$

3.5.1 Symbolic Primitives for Runs

datatype $'\tau$ *constr* =
Timestamp ‹*clock*› ‹*instant-index*› ‹$'\tau$ *tag-const*› $(\text{-} \Downarrow \text{-} @ \text{-})$
| *TimeDelay* ‹*clock*› ‹*instant-index*› ‹$'\tau$ *tag-const*› ‹*clock*› $(\text{-} @ \text{-} \oplus \text{-} \Rightarrow \text{-})$
| *Ticks* ‹*clock*› ‹*instant-index*› $(\text{-} \Uparrow \text{-})$
| *NotTicks* ‹*clock*› ‹*instant-index*› $(\text{-} \neg\Uparrow \text{-})$
| *NotTicksUntil* ‹*clock*› ‹*instant-index*› $(\text{-} \neg\Uparrow < \text{-})$
| *NotTicksFrom* ‹*clock*› ‹*instant-index*› $(\text{-} \neg\Uparrow \geq \text{-})$
| *TagArith* ‹*tag-var*› ‹*tag-var*› ‹$('\tau$ *tag-const* $\times$ $'\tau$ *tag-const*$) \Rightarrow$ *bool*› $(\lfloor\text{-}, \text{-}\rfloor \in \text{-})$
| *TickCntArith* ‹*cnt-expr*› ‹*cnt-expr*› ‹$($*nat* $\times$ *nat*$) \Rightarrow$ *bool*› $(\lceil\text{-}, \text{-}\rceil \in \text{-})$
| *TickCntLeq* ‹*cnt-expr*› ‹*cnt-expr*› $(\text{-} \preceq \text{-})$

type-synonym $'\tau$ *system* = ‹$'\tau$ *constr list*›

— The abstract machine follows the intuition: past [@termΓ], current index [n], present [@termΨ], future [@termΦ] Beware: This type is slightly different from the one originally implemented in Heron
type-synonym $'\tau$ *config* = ‹$'\tau$ *system* $*$ *instant-index* $*$ $'\tau$ *TESL-formula* $*$ $'\tau$ *TESL-formula*›

3.6 Semantics of Primitive Constraints

fun *counter-expr-eval* :: ‹$('\tau$::*linordered-field*$)$ *run* $\Rightarrow$ *cnt-expr* $\Rightarrow$ *nat*› $([\![\text{-} \vdash \text{-}]\!]_{cntexpr})$
where
‹$[\![\varrho \vdash \#^{<}\ clk\ indx]\!]_{cntexpr}$ = *run-tick-count-strictly* ϱ *clk indx*›
| ‹$[\![\varrho \vdash \#^{\leq}\ clk\ indx]\!]_{cntexpr}$ = *run-tick-count* ϱ *clk indx*›

fun *symbolic-run-interpretation-primitive*
::‹$('\tau$::*linordered-field*$)$ *constr* $\Rightarrow$ $'\tau$ *run set*› $([\![\text{-}]\!]_{prim})$
where
‹$[\![K \Uparrow n]\!]_{prim}$ = $\{\varrho.$ *hamlet* ((*Rep-run* ϱ) *n K*) $\}$›
| ‹$[\![K\ @\ n_0 \oplus \delta t \Rightarrow K']\!]_{prim}$ = $\{\varrho.\ \forall n \geq n_0.$ *first-time* ϱ *K n* (*time* ((*Rep-run* ϱ) n_0 *K*) + δt)
$\longrightarrow$ *hamlet* ((*Rep-run* ϱ) *n* K')$\}$›
| ‹$[\![K \neg\Uparrow n]\!]_{prim}$ = $\{\varrho.\ \neg$*hamlet* ((*Rep-run* ϱ) *n K*) $\}$›
| ‹$[\![K \neg\Uparrow < n]\!]_{prim}$ = $\{\varrho.\ \forall i < n.\ \neg$ *hamlet* ((*Rep-run* ϱ) *i K*)$\}$›
| ‹$[\![K \neg\Uparrow \geq n]\!]_{prim}$ = $\{\varrho.\ \forall i \geq n.\ \neg$ *hamlet* ((*Rep-run* ϱ) *i K*) $\}$›

| ‹$[\![K \Downarrow n \mathbin{@} \tau]\!]_{prim} = \{\varrho.\ time\ ((Rep\text{-}run\ \varrho)\ n\ K) = \tau\ \}$›
| ‹$[\![\lfloor \tau_{var}(K_1,\ n_1),\ \tau_{var}(K_2,\ n_2) \rfloor \in R]\!]_{prim} =$
$\{\ \varrho.\ R\ (time\ ((Rep\text{-}run\ \varrho)\ n_1\ K_1),\ time\ ((Rep\text{-}run\ \varrho)\ n_2\ K_2))\ \}$›
| ‹$[\![\lceil e_1,\ e_2 \rceil \in R]\!]_{prim} = \{\ \varrho.\ R\ ([\![\varrho \vdash e_1]\!]_{cntexpr},\ [\![\varrho \vdash e_2]\!]_{cntexpr})\ \}$›
| ‹$[\![cnt\text{-}e_1 \preceq cnt\text{-}e_2]\!]_{prim} = \{\ \varrho.\ [\![\varrho \vdash cnt\text{-}e_1]\!]_{cntexpr} \leq [\![\varrho \vdash cnt\text{-}e_2]\!]_{cntexpr}\ \}$›

fun *symbolic-run-interpretation*
::‹$('\tau$::*linordered-field*) *constr list* $\Rightarrow$ $('\tau$::*linordered-field*) *run set*› ($[\![[\ \text{-}\ ]\!]]_{prim}$)
where
‹$[\![[\ []\ ]\!]]_{prim} = \{\ \text{-}.\ True\ \}$›
| ‹$[\![[\ \gamma \# \Gamma\ ]\!]]_{prim} = [\![\gamma]\!]_{prim} \cap [\![[\ \Gamma\ ]\!]]_{prim}$›

lemma *symbolic-run-interp-cons-morph*:
‹$[\![\gamma]\!]_{prim} \cap [\![[\ \Gamma\ ]\!]]_{prim} = [\![[\ \gamma \# \Gamma\ ]\!]]_{prim}$›
by *auto*

definition *consistent-context* :: ‹$('\tau$::*linordered-field*) *constr list* $\Rightarrow$ *bool*›
where
‹*consistent-context* $\Gamma \equiv \exists\, \varrho.\ \varrho \in [\![[\ \Gamma\ ]\!]]_{prim}$›

3.6.1 Defining a method for witness construction

— Initial states
abbreviation *initial-run* :: ‹$('\tau$::*linordered-field*) *run*› ($\varrho_\odot$) **where**
‹$\varrho_\odot \equiv Abs\text{-}run\ ((\lambda\text{-}\ \text{-}.\ (False,\ \tau_{cst}\ 0))\ ::nat \Rightarrow clock \Rightarrow (bool \times {}'\tau\ tag\text{-}const))$›

— To ensure monotonicity, time tag is set at a specific instant and forever after (stuttering)
fun *time-update*
:: ‹$nat \Rightarrow clock \Rightarrow ('\tau::linordered\text{-}field)\ tag\text{-}const \Rightarrow (nat \Rightarrow clock \Rightarrow (bool \times {}'\tau\ tag\text{-}const))$
$\Rightarrow (nat \Rightarrow clock \Rightarrow (bool \times {}'\tau\ tag\text{-}const))$›
where
‹$time\text{-}update\ n\ K\ \tau\ \varrho = (\lambda n'\ K'.\ if\ K = K' \wedge n \leq n'\ then\ (hamlet\ (\varrho\ n\ K),\ \tau)\ else\ \varrho\ n'\ K')$›

3.7 Rules and properties of consistence

lemma *context-consistency-preservationI*:
‹*consistent-context* $((\gamma :: ('\tau::linordered\text{-}field)\ constr) \# \Gamma) \Longrightarrow$ *consistent-context* Γ›
unfolding *consistent-context-def*
by *auto*

— This is very restrictive
inductive *context-independency*::‹$('\tau$::*linordered-field*) *constr* $\Rightarrow$ $'\tau$ *constr list* $\Rightarrow$ *bool*› (- $\bowtie$ -)
where
NotTicks-independency:

$\langle(K \Uparrow n) \notin$ *set* $\Gamma \Longrightarrow (K \neg\Uparrow n) \bowtie \Gamma\rangle$
| *Ticks-independency*:
$\langle(K \neg\Uparrow n) \notin$ *set* $\Gamma \Longrightarrow (K \Uparrow n) \bowtie \Gamma\rangle$
| *Timestamp-independency*:
$\langle(\nexists\tau'.\ \tau' = \tau \wedge (K \Downarrow n\ @\ \tau) \in$ *set* $\Gamma) \Longrightarrow (K \Downarrow n\ @\ \tau) \bowtie \Gamma\rangle$

~~lemma *context-consistency-preservationE*: assumes *consist*: ‹*consistent-context* Γ› and *indepen*: ‹γ ⋈ Γ› shows ‹*consistent-context* (γ # Γ)› oops~~

3.8 Major Theorems

3.8.1 Fixpoint lemma

theorem *symrun-interp-fixpoint*:
$\langle\bigcap\ ((\lambda\gamma.\ [\![\ \gamma\ ]\!]_{prim})\ `\ set\ \Gamma) = [\![[\ \Gamma\ ]\!]]_{prim}\rangle$
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*:
$\langle[\![[\ \Gamma_1\ @\ \Gamma_2\ ]\!]]_{prim} = [\![[\ \Gamma_1\ ]\!]]_{prim} \cap [\![[\ \Gamma_2\ ]\!]]_{prim}\rangle$
by (*induction* Γ_1, *auto*)

3.9 Equational laws for TESL formulae denotationally interpreted

3.9.1 General laws

lemma *symrun-interp-assoc*:
$\langle[\![[\ (\Gamma_1\ @\ \Gamma_2)\ @\ \Gamma_3\ ]\!]]_{prim} = [\![[\ \Gamma_1\ @\ (\Gamma_2\ @\ \Gamma_3)\ ]\!]]_{prim}\rangle$
by *auto*

lemma *symrun-interp-commute*:
$\langle[\![[\ \Gamma_1\ @\ \Gamma_2\ ]\!]]_{prim} = [\![[\ \Gamma_2\ @\ \Gamma_1\ ]\!]]_{prim}\rangle$
by (*simp add*: *symrun-interp-expansion inf-sup-aci*(*1*))

lemma *symrun-interp-left-commute*:
$\langle[\![[\ \Gamma_1\ @\ (\Gamma_2\ @\ \Gamma_3)\ ]\!]]_{prim} = [\![[\ \Gamma_2\ @\ (\Gamma_1\ @\ \Gamma_3)\ ]\!]]_{prim}\rangle$
unfolding *symrun-interp-expansion* **by** *auto*

lemma *symrun-interp-idem*:
$\langle[\![[\ \Gamma\ @\ \Gamma\ ]\!]]_{prim} = [\![[\ \Gamma\ ]\!]]_{prim}\rangle$
using *symrun-interp-expansion* **by** *auto*

lemma *symrun-interp-left-idem*:
$\langle [\![\Gamma_1 @ (\Gamma_1 @ \Gamma_2)]\!]_{prim} = [\![\Gamma_1 @ \Gamma_2]\!]_{prim} \rangle$
using *symrun-interp-expansion* **by** *auto*

lemma *symrun-interp-right-idem*:
$\langle [\![(\Gamma_1 @ \Gamma_2) @ \Gamma_2]\!]_{prim} = [\![\Gamma_1 @ \Gamma_2]\!]_{prim} \rangle$
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*:
$\langle [\![[] @ \Gamma]\!]_{prim} = [\![\Gamma]\!]_{prim} \rangle$
by *simp*

lemma *symrun-interp-neutral2*:
$\langle [\![\Gamma @ []]\!]_{prim} = [\![\Gamma]\!]_{prim} \rangle$
by *simp*

3.9.2 Decreasing interpretation of TESL formulae

lemma *TESL-sem-decreases-head*:
$\langle [\![\Gamma]\!]_{prim} \supseteq [\![\gamma \# \Gamma]\!]_{prim} \rangle$
by *simp*

lemma *TESL-sem-decreases-tail*:
$\langle [\![\Gamma]\!]_{prim} \supseteq [\![\Gamma @ [\gamma]]\!]_{prim} \rangle$
by (*simp add*: *symrun-interp-expansion*)

lemma *symrun-interp-formula-stuttering*:
assumes $\langle \gamma \in set\ \Gamma \rangle$
shows $\langle [\![\gamma \# \Gamma]\!]_{prim} = [\![\Gamma]\!]_{prim} \rangle$
proof −
have $\langle \gamma \# \Gamma = [\gamma] @ \Gamma \rangle$ **by** *simp*
hence $\langle [\![\gamma \# \Gamma]\!]_{prim} = [\![[\gamma]]\!]_{prim} \cap [\![\Gamma]\!]_{prim} \rangle$ **using** *symrun-interp-expansion*
by *simp*
thus *?thesis* **using** *assms symrun-interp-fixpoint* **by** *fastforce*
qed

lemma *symrun-interp-decreases*:
$\langle [\![\Gamma]\!]_{prim} \supseteq [\![\gamma \# \Gamma]\!]_{prim} \rangle$
by (*rule TESL-sem-decreases-head*)

lemma *symrun-interp-remdups-absorb*:
$\langle [\![\Gamma]\!]_{prim} = [\![remdups\ \Gamma]\!]_{prim} \rangle$

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 ‹*set* Γ = *set* Γ'›
    shows ‹$[\![[\ \Gamma\ ]\!]]_{prim} = [\![[\ \Gamma'\ ]\!]]_{prim}$›
proof −
  have ‹*set* (*remdups* Γ) = *set* (*remdups* Γ')›
    by (*simp add*: *assms*)
  moreover have *fxpnt*Γ: ‹$\bigcap$ (($\lambda\gamma$. $[\![\ \gamma\ ]\!]_{prim}$) ‘ *set* Γ) = $[\![[\ \Gamma\ ]\!]]_{prim}$›
    by (*simp add*: *symrun-interp-fixpoint*)
  moreover have *fxpnt*Γ': ‹$\bigcap$ (($\lambda\gamma$. $[\![\ \gamma\ ]\!]_{prim}$) ‘ *set* Γ') = $[\![[\ \Gamma'\ ]\!]]_{prim}$›
    by (*simp add*: *symrun-interp-fixpoint*)
  moreover have ‹$\bigcap$ (($\lambda\gamma$. $[\![\ \gamma\ ]\!]_{prim}$) ‘ *set* Γ) = $\bigcap$ (($\lambda\gamma$. $[\![\ \gamma\ ]\!]_{prim}$) ‘ *set* Γ')›
    by (*simp add*: *assms*)
  ultimately show *?thesis* **using** *symrun-interp-remdups-absorb* **by** *auto*
qed

theorem *symrun-interp-decreases-setinc*:
  assumes ‹*set* $\Gamma \subseteq$ *set* Γ'›
    shows ‹$[\![[\ \Gamma\ ]\!]]_{prim} \supseteq [\![[\ \Gamma'\ ]\!]]_{prim}$›
proof −
  obtain Γ_r **where** *decompose*: ‹*set* (Γ @ Γ_r) = *set* Γ'› **using** *assms* **by** *auto*
  have ‹*set* (Γ @ Γ_r) = *set* Γ'› **using** *assms decompose* **by** *blast*
  moreover have ‹(*set* Γ) $\cup$ (*set* Γ_r) = *set* Γ'› **using** *assms decompose* **by** *auto*
  moreover have ‹$[\![[\ \Gamma'\ ]\!]]_{prim} = [\![[\ \Gamma\ @\ \Gamma_r\ ]\!]]_{prim}$› **using** *symrun-interp-set-lifting* *decompose* **by** *blast*
  moreover have ‹$[\![[\ \Gamma\ @\ \Gamma_r\ ]\!]]_{prim} = [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Gamma_r\ ]\!]]_{prim}$› **by** (*simp add*: *symrun-interp-expansion*)
  moreover have ‹$[\![[\ \Gamma\ ]\!]]_{prim} \supseteq [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Gamma_r\ ]\!]]_{prim}$› **by** *simp*
  ultimately show *?thesis* **by** *simp*
qed

lemma *symrun-interp-decreases-add-head*:
  assumes ‹*set* $\Gamma \subseteq$ *set* Γ'›
    shows ‹$[\![[\ \gamma\ \#\ \Gamma\ ]\!]]_{prim} \supseteq [\![[\ \gamma\ \#\ \Gamma'\ ]\!]]_{prim}$›
using *symrun-interp-decreases-setinc assms* **by** *auto*

lemma *symrun-interp-decreases-add-tail*:
  assumes ‹*set* $\Gamma \subseteq$ *set* Γ'›
    shows ‹$[\![[\ \Gamma\ @\ [\gamma]\ ]\!]]_{prim} \supseteq [\![[\ \Gamma'\ @\ [\gamma]\ ]\!]]_{prim}$›
proof −
  from *symrun-interp-decreases-setinc*[*OF assms*] **have** ‹$[\![[\ \Gamma'\ ]\!]]_{prim} \subseteq [\![[\ \Gamma\ ]\!]]_{prim}$›
.
  thus *?thesis* **by** (*simp add*: *symrun-interp-expansion dual-order.trans*)

qed

lemma *symrun-interp-absorb1*:
  assumes ‹$set\ \Gamma_1 \subseteq set\ \Gamma_2$›
    shows ‹$[\![[\![\ \Gamma_1\ @\ \Gamma_2\ ]\!]]\!]_{prim} = [\![[\![\ \Gamma_2\ ]\!]]\!]_{prim}$›
by (*simp add*: *Int-absorb1 symrun-interp-decreases-setinc symrun-interp-expansion assms*)

lemma *symrun-interp-absorb2*:
  assumes ‹$set\ \Gamma_2 \subseteq set\ \Gamma_1$›
    shows ‹$[\![[\![\ \Gamma_1\ @\ \Gamma_2\ ]\!]]\!]_{prim} = [\![[\![\ \Gamma_1\ ]\!]]\!]_{prim}$›
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*
$::\langle('\tau::linordered\text{-}field)\ system \Rightarrow instant\text{-}index \Rightarrow '\tau\ TESL\text{-}formula \Rightarrow '\tau\ TESL\text{-}formula \Rightarrow '\tau\ config\rangle$ $(\text{-},\ \text{-} \vdash \text{-} \triangleright \text{-}\ 80)$
where
$\langle\Gamma,\ n \vdash \Psi \triangleright \Phi \equiv (\Gamma,\ n,\ \Psi,\ \Phi)\rangle$

inductive *operational-semantics-intro*
$::\langle('\tau::linordered\text{-}field)\ config \Rightarrow '\tau\ config \Rightarrow bool\rangle$ $(\text{-} \hookrightarrow_i \text{-}\ 70)$
where
instant-i:
$\langle(\Gamma,\ n \vdash [] \triangleright \Phi) \hookrightarrow_i (\Gamma,\ Suc\ n \vdash \Phi \triangleright [])\rangle$

inductive *operational-semantics-elim*
$::\langle('\tau::linordered\text{-}field)\ config \Rightarrow '\tau\ config \Rightarrow bool\rangle$ $(\text{-} \hookrightarrow_e \text{-}\ 70)$
where
sporadic-on-e1:
$\langle(\Gamma,\ n \vdash ((K_1\ sporadic\ \tau\ on\ K_2)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow_e (\Gamma,\ n \vdash \Psi \triangleright ((K_1\ sporadic\ \tau\ on\ K_2)\ \#\ \Phi))\rangle$
| *sporadic-on-e2*:
$\langle(\Gamma,\ n \vdash ((K_1\ sporadic\ \tau\ on\ K_2)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow_e (((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi)\rangle$
| *tagrel-e*:
$\langle(\Gamma,\ n \vdash ((time{-}relation\ \lfloor K_1,\ K_2\rfloor \in R)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow_e (((\lfloor\tau_{var}(K_1,\ n),\ \tau_{var}(K_2,\ n)\rfloor \in R)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((time{-}relation\ \lfloor K_1,$
$K_2\rfloor \in R)\ \#\ \Phi))\rangle$
| *implies-e1*:

‹$(\Gamma, n \vdash ((K_1\ \mathit{implies}\ K_2) \# \Psi) \triangleright \Phi)$
$\hookrightarrow_e\ (((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2) \# \Phi))$›
| *implies-e2*:
‹$(\Gamma, n \vdash ((K_1\ \mathit{implies}\ K_2) \# \Psi) \triangleright \Phi)$
$\hookrightarrow_e\ (((K_1 \Uparrow n) \# (K_2 \Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2) \# \Phi))$›
| *implies-not-e1*:
‹$(\Gamma, n \vdash ((K_1\ \mathit{implies\ not}\ K_2) \# \Psi) \triangleright \Phi)$
$\hookrightarrow_e\ (((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1\ \mathit{implies\ not}\ K_2) \# \Phi))$›
| *implies-not-e2*:
‹$(\Gamma, n \vdash ((K_1\ \mathit{implies\ not}\ K_2) \# \Psi) \triangleright \Phi)$
$\hookrightarrow_e\ (((K_1 \Uparrow n) \# (K_2 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1\ \mathit{implies\ not}\ K_2) \# \Phi))$›
| *timedelayed-e1*:
‹$(\Gamma, n \vdash ((K_1\ \mathit{time{-}delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3) \# \Psi) \triangleright \Phi)$
$\hookrightarrow_e\ (((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1\ \mathit{time{-}delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3)$
$\# \Phi))$›
| *timedelayed-e2*:
‹$(\Gamma, n \vdash ((K_1\ \mathit{time{-}delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3) \# \Psi) \triangleright \Phi)$
$\hookrightarrow_e\ (((K_1 \Uparrow n) \# (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3) \# \Gamma), n$
$\vdash \Psi \triangleright ((K_1\ \mathit{time{-}delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3) \# \Phi))$›
| *weakly-precedes-e*:
‹$(\Gamma, n \vdash ((K_1\ \mathit{weakly\ precedes}\ K_2) \# \Psi) \triangleright \Phi)$
$\hookrightarrow_e\ ((([\#^{\leq}\ K_2\ n, \#^{\leq}\ K_1\ n] \in (\lambda(x,y).\ x \leq y)) \# \Gamma), n$
$\vdash \Psi \triangleright ((K_1\ \mathit{weakly\ precedes}\ K_2) \# \Phi))$›
| *strictly-precedes-e*:
‹$(\Gamma, n \vdash ((K_1\ \mathit{strictly\ precedes}\ K_2) \# \Psi) \triangleright \Phi)$
$\hookrightarrow_e\ ((([\#^{\leq}\ K_2\ n, \#^{<}\ K_1\ n] \in (\lambda(x,y).\ x \leq y)) \# \Gamma), n$
$\vdash \Psi \triangleright ((K_1\ \mathit{strictly\ precedes}\ K_2) \# \Phi))$›
| *kills-e1*:
‹$(\Gamma, n \vdash ((K_1\ \mathit{kills}\ K_2) \# \Psi) \triangleright \Phi)$
$\hookrightarrow_e\ (((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1\ \mathit{kills}\ K_2) \# \Phi))$›
| *kills-e2*:
‹$(\Gamma, n \vdash ((K_1\ \mathit{kills}\ K_2) \# \Psi) \triangleright \Phi)$
$\hookrightarrow_e\ (((K_1 \Uparrow n) \# (K_2 \neg\Uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1\ \mathit{kills}\ K_2) \# \Phi))$›

inductive *operational-semantics-step*
::‹$('\tau{::}\mathit{linordered\text{-}field})\ \mathit{config} \Rightarrow {'\tau}\ \mathit{config} \Rightarrow \mathit{bool}$› (- $\hookrightarrow$ - *70*)
where
intro-part:
‹$(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1)\ \hookrightarrow_i\ (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$
$\Longrightarrow (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1)\ \hookrightarrow\ (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$›
| *elims-part*:
‹$(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1)\ \hookrightarrow_e\ (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$
$\Longrightarrow (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1)\ \hookrightarrow\ (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$›

abbreviation *operational-semantics-step-rtranclp*
::‹$('\tau{::}\mathit{linordered\text{-}field})\ \mathit{config} \Rightarrow {'\tau}\ \mathit{config} \Rightarrow \mathit{bool}$› (- $\hookrightarrow^{**}$ - *70*)
where
‹$\mathcal{C}_1 \hookrightarrow^{**} \mathcal{C}_2 \equiv \mathit{operational\text{-}semantics\text{-}step}^{**}\ \mathcal{C}_1\ \mathcal{C}_2$›

abbreviation *operational-semantics-step-tranclp*
::‹($'\tau$::*linordered-field*) *config* ⇒ $'\tau$ *config* ⇒ *bool*› (- $\hookrightarrow^{++}$ - *70*)
where
‹$\mathcal{C}_1 \hookrightarrow^{++} \mathcal{C}_2 \equiv$ *operational-semantics-step*$^{++}$ $\mathcal{C}_1$ $\mathcal{C}_2$›

abbreviation *operational-semantics-step-reflclp*
::‹($'\tau$::*linordered-field*) *config* ⇒ $'\tau$ *config* ⇒ *bool*› (- $\hookrightarrow^{==}$ - *70*)
where
‹$\mathcal{C}_1 \hookrightarrow^{==} \mathcal{C}_2 \equiv$ *operational-semantics-step*$^{==}$ $\mathcal{C}_1$ $\mathcal{C}_2$›

abbreviation *operational-semantics-step-relpowp*
::‹($'\tau$::*linordered-field*) *config* ⇒ *nat* ⇒ $'\tau$ *config* ⇒ *bool*› (- $\hookrightarrow^{-}$ - *70*)
where
‹$\mathcal{C}_1 \hookrightarrow^{n} \mathcal{C}_2 \equiv$ (*operational-semantics-step* ^^ *n*) $\mathcal{C}_1$ $\mathcal{C}_2$›

definition *operational-semantics-elim-inv*
::‹($'\tau$::*linordered-field*) *config* ⇒ $'\tau$ *config* ⇒ *bool*› (- $\hookrightarrow_e{}^{\leftarrow}$ - *70*)
where
‹$\mathcal{C}_1 \hookrightarrow_e{}^{\leftarrow} \mathcal{C}_2 \equiv \mathcal{C}_2 \hookrightarrow_e \mathcal{C}_1$›

4.2 Basic Lemmas

lemma *operational-semantics-trans-generalized*:
assumes ‹$\mathcal{C}_1 \hookrightarrow^{n} \mathcal{C}_2$›
assumes ‹$\mathcal{C}_2 \hookrightarrow^{m} \mathcal{C}_3$›
shows ‹$\mathcal{C}_1 \hookrightarrow^{n\,+\,m} \mathcal{C}_3$›
using *relcompp.relcompI*[*of* ‹*operational-semantics-step* ^^ *n*› - -
‹*operational-semantics-step* ^^ *m*›, *OF assms*]
by (*simp add*: *relpowp-add*)

abbreviation *Cnext-solve*
::‹($'\tau$::*linordered-field*) *config* ⇒ $'\tau$ *config set*› ($\mathcal{C}_{next}$ -)
where
‹$\mathcal{C}_{next}\ \mathcal{S} \equiv \{\ \mathcal{S}'.\ \mathcal{S} \hookrightarrow \mathcal{S}'\ \}$›

lemma *Cnext-solve-instant*:
‹$(\mathcal{C}_{next}\ (\Gamma,\ n \vdash [] \triangleright \Phi)) \supseteq \{\ \Gamma,\ Suc\ n \vdash \Phi \triangleright []\ \}$›
by (*simp add*: *operational-semantics-step.simps operational-semantics-intro.instant-i*)

lemma *Cnext-solve-sporadicon*:
‹$(\mathcal{C}_{next}\ (\Gamma,\ n \vdash ((K_1\ sporadic\ \tau\ on\ K_2)\ \#\ \Psi) \triangleright \Phi))$
$\supseteq \{\ \Gamma,\ n \vdash \Psi \triangleright ((K_1\ sporadic\ \tau\ on\ K_2)\ \#\ \Phi),\ ((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#$
$\Gamma),\ n \vdash \Psi \triangleright \Phi\ \}$›
by (*simp add*: *operational-semantics-step.simps operational-semantics-elim.sporadic-on-e1*
operational-semantics-elim.sporadic-on-e2)

lemma *Cnext-solve-tagrel*:
‹$(\mathcal{C}_{next}\ (\Gamma,\ n \vdash ((time{-}relation\ \lfloor K_1,\ K_2\rfloor \in R)\ \#\ \Psi) \triangleright \Phi))$
$\supseteq \{\ ((\lfloor \tau_{var}(K_1,\ n),\ \tau_{var}(K_2,\ n)\rfloor \in R)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((time{-}relation\ \lfloor K_1,$

$K_2\rfloor \in R)$ # $\Phi)$ }›
by (*simp add*: *operational-semantics-step.simps operational-semantics-elim.tagrel-e*)

lemma *Cnext-solve-implies*:
‹($\mathcal{C}_{next}$ (Γ, $n \vdash$ ((K_1 *implies* K_2) # Ψ) $\triangleright$ Φ))
$\supseteq$ { ((K_1 $\neg\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies* K_2) # Φ),
((K_1 $\Uparrow$ n) # (K_2 $\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies* K_2) # Φ) }›
by (*simp add*: *operational-semantics-step.simps operational-semantics-elim.implies-e1 operational-semantics-elim.implies-e2*)

lemma *Cnext-solve-implies-not*:
‹($\mathcal{C}_{next}$ (Γ, $n \vdash$ ((K_1 *implies not* K_2) # Ψ) $\triangleright$ Φ))
$\supseteq$ { ((K_1 $\neg\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies not* K_2) # Φ),
((K_1 $\Uparrow$ n) # (K_2 $\neg\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies not* K_2) # Φ) }›
by (*simp add*: *operational-semantics-step.simps operational-semantics-elim.implies-not-e1 operational-semantics-elim.implies-not-e2*)

lemma *Cnext-solve-timedelayed*:
‹($\mathcal{C}_{next}$ (Γ, $n \vdash$ ((K_1 *time−delayed by* $\delta\tau$ *on* K_2 *implies* K_3) # Ψ) $\triangleright$ Φ))
$\supseteq$ { ((K_1 $\neg\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *time−delayed by* $\delta\tau$ *on* K_2 *implies* K_3) # Φ),
((K_1 $\Uparrow$ n) # (K_2 @ $n \oplus \delta\tau \Rightarrow K_3$) # Γ), n
$\vdash \Psi \triangleright$ ((K_1 *time−delayed by* $\delta\tau$ *on* K_2 *implies* K_3) # Φ) }›
by (*simp add*: *operational-semantics-step.simps operational-semantics-elim.timedelayed-e1 operational-semantics-elim.timedelayed-e2*)

lemma *Cnext-solve-weakly-precedes*:
‹($\mathcal{C}_{next}$ (Γ, $n \vdash$ ((K_1 *weakly precedes* K_2) # Ψ) $\triangleright$ Φ))
$\supseteq$ { (($\lceil\#^{\leq}$ K_2 n, $\#^{\leq}$ K_1 $n\rceil \in (\lambda(x,y).\ x \leq y)$) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *weakly precedes* K_2) # Φ) }›
by (*simp add*: *operational-semantics-step.simps operational-semantics-elim.weakly-precedes-e*)

lemma *Cnext-solve-strictly-precedes*:
‹($\mathcal{C}_{next}$ (Γ, $n \vdash$ ((K_1 *strictly precedes* K_2) # Ψ) $\triangleright$ Φ))
$\supseteq$ { (($\lceil\#^{\leq}$ K_2 n, $\#^{<}$ K_1 $n\rceil \in (\lambda(x,y).\ x \leq y)$) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *strictly precedes* K_2) # Φ) }›
by (*simp add*: *operational-semantics-step.simps operational-semantics-elim.strictly-precedes-e*)

lemma *Cnext-solve-kills*:
‹($\mathcal{C}_{next}$ (Γ, $n \vdash$ ((K_1 *kills* K_2) # Ψ) $\triangleright$ Φ))
$\supseteq$ { ((K_1 $\neg\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *kills* K_2) # Φ),
((K_1 $\Uparrow$ n) # (K_2 $\neg\Uparrow \geq n$) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *kills* K_2) # Φ) }›
by (*simp add*: *operational-semantics-step.simps operational-semantics-elim.kills-e1 operational-semantics-elim.kills-e2*)

lemma *empty-spec-reductions*:
‹([], *0* $\vdash$ [] $\triangleright$ []) $\hookrightarrow^k$ ([], $k \vdash$ [] $\triangleright$ [])›
proof (*induct k*)
case *0* **thus** *?case* **by** *simp*

next
case *Suc* **thus** *?case*
using *instant-i operational-semantics-step.simps* **by** *fastforce*
qed

end

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

$\longrightarrow$ *hamlet* ((*Rep-run* ϱ) *m slave*)
)
}›
| ‹⟦ K_1 *weakly precedes* K_2 ⟧$_{TESL}$$^{\geq\ i}$ =
{ ϱ. $\forall\, n \geq i$. (*run-tick-count* ϱ K_2 *n*) $\leq$ (*run-tick-count* ϱ K_1 *n*) }›
| ‹⟦ K_1 *strictly precedes* K_2 ⟧$_{TESL}$$^{\geq\ i}$ =
{ ϱ. $\forall\, n \geq i$. (*run-tick-count* ϱ K_2 *n*) $\leq$ (*run-tick-count-strictly* ϱ K_1 *n*) }›
| ‹⟦ K_1 *kills* K_2 ⟧$_{TESL}$$^{\geq\ i}$ =
{ ϱ. $\forall\, n \geq i$. *hamlet* ((*Rep-run* ϱ) *n* K_1) $\longrightarrow$ ($\forall\, m \geq n$. $\neg$ *hamlet* ((*Rep-run* ϱ) *m* K_2)) }›

theorem *predicate-Inter-unfold*:
‹{ ϱ. $\forall\, n$. *P* ϱ *n*} = $\bigcap$ {*Y*. $\exists\, n$. *Y* = { ϱ. *P* ϱ *n* }}›
by (*simp add*: *Collect-all-eq full-SetCompr-eq*)

theorem *predicate-Union-unfold*:
‹{ ϱ. $\exists\, n$. *P* ϱ *n*} = $\bigcup$ {*Y*. $\exists\, n$. *Y* = { ϱ. *P* ϱ *n* }}›
by *auto*

lemma *TESL-interp-unfold-stepwise-sporadicon*:
shows ‹⟦ K_1 *sporadic* τ *on* K_2 ⟧$_{TESL}$ = $\bigcup$ {*Y*. $\exists\, n$::*nat*. *Y* = ⟦ K_1 *sporadic* τ *on* K_2 ⟧$_{TESL}$$^{\geq\ n}$}›
by *auto*

lemma *TESL-interp-unfold-stepwise-tagrelgen*:
shows ‹⟦ *time−relation* ⌊K_1, K_2⌋ $\in$ *R* ⟧$_{TESL}$ = $\bigcap$ {*Y*. $\exists\, n$::*nat*. *Y* = ⟦ *time−relation* ⌊K_1, K_2⌋ $\in$ *R* ⟧$_{TESL}$$^{\geq\ n}$}›
by *auto*

lemma *TESL-interp-unfold-stepwise-implies*:
shows ‹⟦ *master implies slave* ⟧$_{TESL}$ = $\bigcap$ {*Y*. $\exists\, n$::*nat*. *Y* = ⟦ *master implies slave* ⟧$_{TESL}$$^{\geq\ n}$}›
by *auto*

lemma *TESL-interp-unfold-stepwise-implies-not*:
shows ‹⟦ *master implies not slave* ⟧$_{TESL}$ = $\bigcap$ {*Y*. $\exists\, n$::*nat*. *Y* = ⟦ *master implies not slave* ⟧$_{TESL}$$^{\geq\ n}$}›
by *auto*

lemma *TESL-interp-unfold-stepwise-timedelayed*:
shows ‹⟦ *master time−delayed by* $\delta\tau$ *on measuring implies slave* ⟧$_{TESL}$
= $\bigcap$ {*Y*. $\exists\, n$::*nat*. *Y* = ⟦ *master time−delayed by* $\delta\tau$ *on measuring implies slave* ⟧$_{TESL}$$^{\geq\ n}$}›
by *auto*

lemma *TESL-interp-unfold-stepwise-weakly-precedes*:
shows ‹⟦ K_1 *weakly precedes* K_2 ⟧$_{TESL}$ = $\bigcap$ {*Y*. $\exists\, n$::*nat*. *Y* = ⟦ K_1 *weakly precedes* K_2 ⟧$_{TESL}$$^{\geq\ n}$}›

by *auto*

lemma *TESL-interp-unfold-stepwise-strictly-precedes*:
shows ‹$[\![$ K_1 *strictly precedes* K_2 $]\!]_{TESL} = \bigcap \{Y.\ \exists\, n{::}nat.\ Y = [\![$ K_1 *strictly precedes* K_2 $]\!]_{TESL}{}^{\geq\ n}\}$›
by *auto*

lemma *TESL-interp-unfold-stepwise-kills*:
shows ‹$[\![$ *master kills slave* $]\!]_{TESL} = \bigcap \{Y.\ \exists\, n{::}nat.\ Y = [\![$ *master kills slave* $]\!]_{TESL}{}^{\geq\ n}\}$›
by *auto*

theorem *TESL-interp-unfold-stepwise-positive-atoms*:
assumes ‹*positive-atom* φ›
shows ‹$[\![$ $\varphi{::}'\tau{::}$*linordered-field TESL-atomic* $]\!]_{TESL} = \bigcup \{Y.\ \exists\, n{::}nat.\ Y = [\![\ \varphi\ ]\!]_{TESL}{}^{\geq\ n}\}$›
proof −
from *positive-atom.elims(2)[OF assms]*
obtain *u v w* **where** ‹φ = (*u sporadic v on w*)› **by** *blast*
with *TESL-interp-unfold-stepwise-sporadicon* **show** *?thesis* **by** *simp*
qed

theorem *TESL-interp-unfold-stepwise-negative-atoms*:
assumes ‹¬ *positive-atom* φ›
shows ‹$[\![\ \varphi\ ]\!]_{TESL} = \bigcap \{Y.\ \exists\, n{::}nat.\ Y = [\![\ \varphi\ ]\!]_{TESL}{}^{\geq\ n}\}$›
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*:
‹$(\forall n \geq (n_0::nat).\ P\ n) = (P\ n_0 \wedge (\forall n \geq Suc\ n_0.\ P\ n))$›
proof −
have ‹$(\forall n \geq (n_0::nat).\ P\ n) = (\forall n.\ (n = n_0 \vee n > n_0) \longrightarrow P\ n)$› **using** *le-less* **by** *blast*
also have ‹$... = (P\ n_0 \wedge (\forall n > n_0.\ P\ n))$› **by** *blast*
finally show *?thesis* **using** *Suc-le-eq* **by** *simp*
qed

lemma *exists-nat-expansion*:
‹$(\exists n \geq (n_0::nat).\ P\ n) = (P\ n_0 \vee (\exists n \geq Suc\ n_0.\ P\ n))$›
proof −
have ‹$(\exists n \geq (n_0::nat).\ P\ n) = (\exists n.\ (n = n_0 \vee n > n_0) \wedge P\ n)$› **using** *le-less* **by** *blast*
also have $... = (\exists n.\ (P\ n_0) \vee (n > n_0 \wedge P\ n))$ **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 ‹$[\![\ K_1\ sporadic\ \tau\ on\ K_2\ ]\!]_{TESL}{}^{\geq\ n} =$
$[\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2 \Downarrow n\ @\ \tau\ ]\!]_{prim}$
$\cup\ [\![\ K_1\ sporadic\ \tau\ on\ K_2\ ]\!]_{TESL}{}^{\geq\ Suc\ n}$›
proof −
have ‹$\{\ \varrho.\ \exists m{\geq}n.\ hamlet\ ((Rep\text{-}run\ \varrho)\ m\ K_1) = True \wedge time\ ((Rep\text{-}run\ \varrho)\ m\ K_2) = \tau\ \}$
$= \{\ \varrho.\ hamlet\ ((Rep\text{-}run\ \varrho)\ n\ K_1) = True \wedge time\ ((Rep\text{-}run\ \varrho)\ n\ K_2) = \tau$
$\vee\ (\exists m{\geq}Suc\ n.\ hamlet\ ((Rep\text{-}run\ \varrho)\ m\ K_1) = True \wedge time\ ((Rep\text{-}run\ \varrho)\ m\ K_2) = \tau)\ \}$›
using *Suc-leD not-less-eq-eq* **by** *fastforce*
moreover have ‹$\{\ \varrho.\ hamlet\ ((Rep\text{-}run\ \varrho)\ n\ K_1) = True \wedge time\ ((Rep\text{-}run\ \varrho)\ n\ K_2) = \tau$
$\vee\ (\exists m{\geq}Suc\ n.\ hamlet\ ((Rep\text{-}run\ \varrho)\ m\ K_1) = True \wedge time\ ((Rep\text{-}run\ \varrho)\ m\ K_2) = \tau)\ \}$
$= [\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2 \Downarrow n\ @\ \tau\ ]\!]_{prim} \cup [\![\ K_1\ sporadic\ \tau\ on\ K_2\ ]\!]_{TESL}{}^{\geq\ Suc\ n}$›
by (*simp add*: *Collect-conj-eq Collect-disj-eq*)
ultimately show *?thesis* **by** *auto*
qed

~~*lemma TESL-interp-stepwise-sporadicon-add-coind-unfold*: *shows*~~ $\cancel{‹[\![\ K_1\ sporadic\ (\!(\tau_{var}(K,\ n'') \oplus \tau)\!)\ on\ K_2\ ]\!]_{TESL}{}^{\geq\ n} = [\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2 \Downarrow n\ @\ (\!(\tau_{var}(K,\ n'') \oplus \tau)\!)\ ]\!]_{prim} \cup [\![\ K_1\ sporadic\ (\!(\tau_{var}(K,\ n'') \oplus \tau)\!)\ on\ K_2\ ]\!]_{TESL}{}^{\geq\ Suc\ n}›}$ ~~*proof*~~

~~have ‹{ ϱ. ∃m≥n. hamlet ((Rep-run ϱ) m K_1) = True ∧ time ((Rep-run ϱ) m K_2) = time ((Rep-run ϱ) n'' K) + τ } = { ϱ. hamlet ((Rep-run ϱ) n K_1) = True ∧ time ((Rep-run ϱ) n K_2) = time ((Rep-run ϱ) n'' K) + τ ∨ (∃m≥Suc n. hamlet ((Rep-run ϱ) m K_1) = True ∧ time ((Rep-run ϱ) m K_2) = time ((Rep-run ϱ) n'' K) + τ) }› using Suc-leD not-less-eq-eq by fastforce then show ?thesis by auto qed~~

lemma *TESL-interp-stepwise-sporadicon-coind-unfold*:
 shows ‹$[\![\ K_1\ \textit{sporadic}\ \tau\ \textit{on}\ K_2\ ]\!]_{TESL}{}^{\geq\ n} =$
  $[\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2 \Downarrow n\ @\ \tau\ ]\!]_{prim}$
  $\cup\ [\![\ K_1\ \textit{sporadic}\ \tau\ \textit{on}\ K_2\ ]\!]_{TESL}{}^{\geq\ \textit{Suc}\ n}$›
 using *TESL-interp-stepwise-sporadicon-cst-coind-unfold* **by** *blast*

lemma *nat-set-suc*:‹$\{x.\ \forall m \geq n.\ P\ x\ m\} = \{x.\ P\ x\ n\} \cap \{x.\ \forall m \geq \textit{Suc}\ n.\ P\ x\ m\}$›
proof
 { fix x
  assume *h*:‹$x \in \{x.\ \forall m \geq n.\ P\ x\ m\}$›
  hence ‹$P\ x\ n$› **by** *simp*
  moreover from *h* **have** ‹$x \in \{x.\ \forall m \geq \textit{Suc}\ n.\ P\ x\ m\}$› **by** *simp*
  ultimately have ‹$x \in \{x.\ P\ x\ n\} \cap \{x.\ \forall m \geq \textit{Suc}\ n.\ P\ x\ m\}$› **by** *simp*
 } thus ‹$\{x.\ \forall m \geq n.\ P\ x\ m\} \subseteq \{x.\ P\ x\ n\} \cap \{x.\ \forall m \geq \textit{Suc}\ n.\ P\ x\ m\}$› **..**
next
 { fix x
  assume *h*:‹$x \in \{x.\ P\ x\ n\} \cap \{x.\ \forall m \geq \textit{Suc}\ n.\ P\ x\ m\}$›
  hence ‹$P\ x\ n$› **by** *simp*
  moreover from *h* **have** ‹$\forall m \geq \textit{Suc}\ n.\ P\ x\ m$› **by** *simp*
  ultimately have ‹$\forall m \geq n.\ P\ x\ m$› **using** *forall-nat-expansion* **by** *blast*
  hence ‹$x \in \{x.\ \forall m \geq n.\ P\ x\ m\}$› **by** *simp*
 } thus ‹$\{x.\ P\ x\ n\} \cap \{x.\ \forall m \geq \textit{Suc}\ n.\ P\ x\ m\} \subseteq \{x.\ \forall m \geq n.\ P\ x\ m\}$› **..**
qed

lemma *TESL-interp-stepwise-tagrel-coind-unfold*:
 shows ‹$[\![\ \textit{time-relation}\ \lfloor K_1,\ K_2 \rfloor \in R\ ]\!]_{TESL}{}^{\geq\ n} =$
  $[\![\ \lfloor \tau_{var}(K_1,\ n),\ \tau_{var}(K_2,\ n) \rfloor \in R\ ]\!]_{prim}$
  $\cap\ [\![\ \textit{time-relation}\ \lfloor K_1,\ K_2 \rfloor \in R\ ]\!]_{TESL}{}^{\geq\ \textit{Suc}\ n}$›
 proof –
  have ‹{ ϱ. ∀m≥n. R (*time* ((*Rep-run* ϱ) *m* K_1), *time* ((*Rep-run* ϱ) *m* K_2)) }
    = { ϱ. R (*time* ((*Rep-run* ϱ) *n* K_1), *time* ((*Rep-run* ϱ) *n* K_2)) }
   ∩ { ϱ. ∀m≥*Suc n*. R (*time* ((*Rep-run* ϱ) *m* K_1), *time* ((*Rep-run* ϱ) *m* K_2))
}›
   using *nat-set-suc*[*of* ‹*n*› ‹λ*x y*. *R* (*time* ((*Rep-run x*) *y* K_1), *time* ((*Rep-run x*) *y* K_2))›] **by** *simp*
 then show *?thesis* **by** *auto*
qed

lemma *TESL-interp-stepwise-implies-coind-unfold*:
 shows ‹$[\![\ \textit{master implies slave}\ ]\!]_{TESL}{}^{\geq\ n} =$
  $([\![\ \textit{master} \neg\Uparrow n\ ]\!]_{prim} \cup [\![\ \textit{master} \Uparrow n\ ]\!]_{prim} \cap [\![\ \textit{slave} \Uparrow n\ ]\!]_{prim})$

∩ ⟦ *master implies slave* ⟧$_{TESL}$$^{\geq\ Suc\ n}$›
proof −
have ‹{ ϱ. ∀ *m*≥*n*. *hamlet* ((*Rep-run* ϱ) *m master*) ⟶ *hamlet* ((*Rep-run* ϱ) *m slave*) }
= { ϱ. *hamlet* ((*Rep-run* ϱ) *n master*) ⟶ *hamlet* ((*Rep-run* ϱ) *n slave*) }
∩ { ϱ. ∀ *m*≥*Suc n*. *hamlet* ((*Rep-run* ϱ) *m master*) ⟶ *hamlet* ((*Rep-run* ϱ) *m slave*) }›
using *nat-set-suc*[*of* ‹*n*› ‹λ*x y*. *hamlet* ((*Rep-run x*) *y master*) ⟶ *hamlet* ((*Rep-run x*) *y slave*)›] **by** *simp*
then show *?thesis* **by** *auto*
qed

lemma *TESL-interp-stepwise-implies-not-coind-unfold*:
shows ‹⟦ *master implies not slave* ⟧$_{TESL}$$^{\geq\ n}$ =
(⟦ *master* ¬⇑ *n* ⟧$_{prim}$ ∪ ⟦ *master* ⇑ *n* ⟧$_{prim}$ ∩ ⟦ *slave* ¬⇑ *n* ⟧$_{prim}$)
∩ ⟦ *master implies not slave* ⟧$_{TESL}$$^{\geq\ Suc\ n}$›
proof −
have ‹{ ϱ. ∀ *m*≥*n*. *hamlet* ((*Rep-run* ϱ) *m master*) ⟶ ¬ *hamlet* ((*Rep-run* ϱ) *m slave*) }
= { ϱ. *hamlet* ((*Rep-run* ϱ) *n master*) ⟶ ¬ *hamlet* ((*Rep-run* ϱ) *n slave*) }
∩ { ϱ. ∀ *m*≥*Suc n*. *hamlet* ((*Rep-run* ϱ) *m master*) ⟶ ¬ *hamlet* ((*Rep-run* ϱ) *m slave*) }›
using *nat-set-suc*[*of* ‹*n*› ‹λ*x y*. *hamlet* ((*Rep-run x*) *y master*) ⟶ ¬ *hamlet* ((*Rep-run x*) *y slave*)›] **by** *simp*
then show *?thesis* **by** *auto*
qed

lemma *TESL-interp-stepwise-timedelayed-coind-unfold*:
shows ‹⟦ *master time−delayed by* δτ *on measuring implies slave* ⟧$_{TESL}$$^{\geq\ n}$ =
(⟦ *master* ¬⇑ *n* ⟧$_{prim}$ ∪ (⟦ *master* ⇑ *n* ⟧$_{prim}$ ∩ ⟦ *measuring* @ *n* ⊕ δτ ⇒ *slave* ⟧$_{prim}$))
∩ ⟦ *master time−delayed by* δτ *on measuring implies slave* ⟧$_{TESL}$$^{\geq\ Suc\ n}$›
proof −
let *?prop* = ‹λϱ *m*. *hamlet* ((*Rep-run* ϱ) *m master*) ⟶
(*let measured-time* = *time* ((*Rep-run* ϱ) *m measuring*) *in*
∀ *p* ≥ *m*. *first-time* ϱ *measuring p* (*measured-time* + δτ)
⟶ *hamlet* ((*Rep-run* ϱ) *p slave*))›
have ‹{ ϱ. ∀ *m* ≥ *n*. *?prop* ϱ *m*} = { ϱ. *?prop* ϱ *n*} ∩ {ϱ. ∀ *m* ≥ *Suc n*. *?prop* ϱ *m*}›
using *nat-set-suc*[*of* ‹*n*› *?prop*] **by** *blast*
also have ‹... = { ϱ. *?prop* ϱ *n* } ∩ ⟦ *master time−delayed by* δτ *on measuring implies slave* ⟧$_{TESL}$$^{\geq\ Suc\ n}$› **by** *simp*
finally show *?thesis* **by** *auto*
qed

lemma *TESL-interp-stepwise-weakly-precedes-coind-unfold*:
shows ‹⟦ K_1 *weakly precedes* K_2 ⟧$_{TESL}$$^{\geq\ n}$ =
⟦ (⌈#$^{\leq}$ K_2 *n*, #$^{\leq}$ K_1 *n*⌉ ∈ (λ(*x*,*y*). *x*≤*y*)) ⟧$_{prim}$

$\cap$ $[\![$ K_1 *weakly precedes* K_2 $]\!]_{TESL}{}^{\geq\ Suc\ n}$›
proof –
have ‹{ ϱ. $\forall p{\geq}n$. (*run-tick-count* ϱ K_2 p) $\leq$ (*run-tick-count* ϱ K_1 p) }
= { ϱ. (*run-tick-count* ϱ K_2 n) $\leq$ (*run-tick-count* ϱ K_1 n) }
$\cap$ { ϱ. $\forall p{\geq}Suc\ n$. (*run-tick-count* ϱ K_2 p) $\leq$ (*run-tick-count* ϱ K_1 p) }›
using *nat-set-suc*[*of* ‹n› ‹$\lambda\varrho$ n. (*run-tick-count* ϱ K_2 n) $\leq$ (*run-tick-count* ϱ K_1 n)›]
by *simp*
then show *?thesis* **by** *auto*
qed

lemma *TESL-interp-stepwise-strictly-precedes-coind-unfold*:
shows ‹$[\![$ K_1 *strictly precedes* K_2 $]\!]_{TESL}{}^{\geq\ n}$ =
$[\![$ ($\lceil\#^{\leq}$ K_2 n, $\#^{<}$ K_1 $n\rceil$ $\in$ ($\lambda(x,y)$. $x{\leq}y$)) $]\!]_{prim}$
$\cap$ $[\![$ K_1 *strictly precedes* K_2 $]\!]_{TESL}{}^{\geq\ Suc\ n}$›
proof –
have ‹{ ϱ. $\forall p{\geq}n$. (*run-tick-count* ϱ K_2 p) $\leq$ (*run-tick-count-strictly* ϱ K_1 p) }
= { ϱ. (*run-tick-count* ϱ K_2 n) $\leq$ (*run-tick-count-strictly* ϱ K_1 n) }
$\cap$ { ϱ. $\forall p{\geq}Suc\ n$. (*run-tick-count* ϱ K_2 p) $\leq$ (*run-tick-count-strictly* ϱ K_1 p) }›
using *nat-set-suc*[*of* ‹n› ‹$\lambda\varrho$ n. (*run-tick-count* ϱ K_2 n) $\leq$ (*run-tick-count-strictly* ϱ K_1 n)›]
by *simp*
then show *?thesis* **by** *auto*
qed

lemma *TESL-interp-stepwise-kills-coind-unfold*:
shows ‹$[\![$ K_1 *kills* K_2 $]\!]_{TESL}{}^{\geq\ n}$ =
($[\![$ K_1 $\neg\Uparrow$ n $]\!]_{prim}$ $\cup$ $[\![$ K_1 $\Uparrow$ n $]\!]_{prim}$ $\cap$ $[\![$ K_2 $\neg\Uparrow$ $\geq$ n $]\!]_{prim}$)
$\cap$ $[\![$ K_1 *kills* K_2 $]\!]_{TESL}{}^{\geq\ Suc\ n}$›
proof –
let *?kills* = ‹λn ϱ. $\forall p{\geq}n$. *hamlet* ((*Rep-run* ϱ) p K_1) $\longrightarrow$ ($\forall m{\geq}p$. $\neg$ *hamlet* ((*Rep-run* ϱ) m K_2))›
let *?ticks* = ‹λn ϱ c. *hamlet* ((*Rep-run* ϱ) n c)›
let *?dead* = ‹λn ϱ c. $\forall m \geq n$. $\neg$*hamlet* ((*Rep-run* ϱ) m c)›
have ‹$[\![$ K_1 *kills* K_2 $]\!]_{TESL}{}^{\geq\ n}$ = {ϱ. *?kills* n ϱ}› **by** *simp*
also have ‹... = ({ϱ. $\neg$ *?ticks* n ϱ K_1} $\cap$ { ϱ. *?kills* (*Suc* n) ϱ})
$\cup$ ({ϱ. *?ticks* n ϱ K_1} $\cap$ {ϱ. *?dead* n ϱ K_2})›
proof
{ fix ϱ::‹$'\tau$::*linordered-field run*›
assume ‹$\varrho \in$ {ϱ. *?kills* n ϱ}›
hence ‹*?kills* n ϱ› **by** *simp*
hence ‹(*?ticks* n ϱ K_1 $\wedge$ *?dead* n ϱ K_2) $\vee$ ($\neg$*?ticks* n ϱ K_1 $\wedge$ *?kills* (*Suc* n) ϱ)›
using *Suc-leD* **by** *blast*
hence ‹$\varrho \in$ ({ϱ. *?ticks* n ϱ K_1} $\cap$ {ϱ. *?dead* n ϱ K_2})
$\cup$ ({ϱ. $\neg$ *?ticks* n ϱ K_1} $\cap$ {ϱ. *?kills* (*Suc* n) ϱ})›
by *blast*
} thus ‹{ϱ. *?kills* n ϱ}

$\subseteq \{\varrho.\ \neg\ ?ticks\ n\ \varrho\ K_1\} \cap \{\varrho.\ ?kills\ (Suc\ n)\ \varrho\}$
$\cup \{\varrho.\ ?ticks\ n\ \varrho\ K_1\} \cap \{\varrho.\ ?dead\ n\ \varrho\ K_2\}\rangle$ **by** *blast*
next
{ fix ϱ::⟨$'\tau$::*linordered-field run*⟩
assume ⟨$\varrho \in (\{\varrho.\ \neg\ ?ticks\ n\ \varrho\ K_1\} \cap \{\varrho.\ ?kills\ (Suc\ n)\ \varrho\})$
$\cup (\{\varrho.\ ?ticks\ n\ \varrho\ K_1\} \cap \{\varrho.\ ?dead\ n\ \varrho\ K_2\})$⟩
hence ⟨$\neg\ ?ticks\ n\ \varrho\ K_1 \wedge ?kills\ (Suc\ n)\ \varrho$
$\vee\ ?ticks\ n\ \varrho\ K_1 \wedge ?dead\ n\ \varrho\ K_2$⟩ **by** *blast*
moreover have ⟨$((\neg\ ?ticks\ n\ \varrho\ K_1) \wedge (?kills\ (Suc\ n)\ \varrho)) \longrightarrow ?kills\ n\ \varrho$⟩
using *dual-order.antisym not-less-eq-eq* **by** *blast*
ultimately have ⟨$?kills\ n\ \varrho \vee ?ticks\ n\ \varrho\ K_1 \wedge ?dead\ n\ \varrho\ K_2$⟩ **by** *blast*
hence ⟨$?kills\ n\ \varrho$⟩ **using** *le-trans* **by** *blast*
} thus ⟨$(\{\varrho.\ \neg\ ?ticks\ n\ \varrho\ K_1\} \cap \{\varrho.\ ?kills\ (Suc\ n)\ \varrho\})$
$\cup (\{\varrho.\ ?ticks\ n\ \varrho\ K_1\} \cap \{\varrho.\ ?dead\ n\ \varrho\ K_2\})$
$\subseteq \{\varrho.\ ?kills\ n\ \varrho\}$⟩ **by** *blast*
qed
also have ⟨$... = \{\varrho.\ \neg\ ?ticks\ n\ \varrho\ K_1\} \cap \{\varrho.\ ?kills\ (Suc\ n)\ \varrho\}$
$\cup \{\varrho.\ ?ticks\ n\ \varrho\ K_1\} \cap \{\varrho.\ ?dead\ n\ \varrho\ K_2\} \cap \{\varrho.\ ?kills\ (Suc\ n)\ \varrho\}$⟩
using *Collect-cong Collect-disj-eq* **by** *auto*
also have ⟨$... = [\![\ K_1 \neg\Uparrow n\ ]\!]_{prim} \cap [\![\ K_1\ kills\ K_2\ ]\!]_{TESL}{}^{\geq\ Suc\ n}$
$\cup [\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2 \neg\Uparrow \geq n\ ]\!]_{prim} \cap [\![\ K_1\ kills\ K_2\ ]\!]_{TESL}{}^{\geq\ Suc\ n}$⟩
by *simp*
finally show *?thesis* **by** *blast*
qed

fun *TESL-interpretation-stepwise* :: ⟨$'\tau$::*linordered-field TESL-formula* $\Rightarrow$ *nat* $\Rightarrow$ $'\tau$ *run set*⟩ ($[\![[\ \text{-}\ ]\!]]_{TESL}{}^{\geq\ \text{-}}$) **where**
⟨$[\![[\ [\,]\ ]\!]]_{TESL}{}^{\geq\ n} = \{\ \text{-}.\ True\ \}$⟩
| ⟨$[\![[\ \varphi\ \#\ \Phi\ ]\!]]_{TESL}{}^{\geq\ n} = [\![\ \varphi\ ]\!]_{TESL}{}^{\geq\ n} \cap [\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ n}$⟩

lemma *TESL-interpretation-stepwise-fixpoint*:
⟨$[\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ n} = \bigcap\ ((\lambda\varphi.\ [\![\ \varphi\ ]\!]_{TESL}{}^{\geq\ n})\ \text{`}\ set\ \Phi)$⟩
proof (*induct* Φ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a* Φ)
then show *?case* **by** *auto*
qed

lemma *TESL-interpretation-stepwise-zero*:
⟨$[\![\ \varphi\ ]\!]_{TESL} = [\![\ \varphi\ ]\!]_{TESL}{}^{\geq\ 0}$⟩
proof (*induct* φ)
case (*SporadicOn* $K_1\ \tau\ K_2$)
then show *?case* **by** *simp*
next
case (*TagRelation x1 x2 x3*)
then show *?case* **by** *simp*
next

case (*Implies x1 x2*)
then show *?case* **by** *simp*
next
case (*ImpliesNot x1 x2*)
then show *?case* **by** *simp*
next
case (*TimeDelayedBy x1 x2 x3 x4*)
then show *?case* **by** *simp*
next
case (*WeaklyPrecedes x1 x2*)
then show *?case* **by** *simp*
next
case (*StrictlyPrecedes x1 x2*)
then show *?case* **by** *simp*
next
case (*Kills x1 x2*)
then show *?case* **by** *simp*
qed

lemma *TESL-interpretation-stepwise-zero'*:
$\langle [\![[\![\Phi]\!]]\!]_{TESL} = [\![[\![\Phi]\!]]\!]_{TESL}^{\geq\ 0} \rangle$
proof (*induct* Φ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a* Φ)
then show *?case*
by (*simp add*: *TESL-interpretation-stepwise-zero*)
qed

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

theorem *TESL-interp-stepwise-composition*:
shows $\langle [\![[\![\Phi_1 @ \Phi_2]\!]]\!]_{TESL}^{\geq\ n} = [\![[\![\Phi_1]\!]]\!]_{TESL}^{\geq\ n} \cap [\![[\![\Phi_2]\!]]\!]_{TESL}^{\geq\ n} \rangle$
proof (*induct* Φ_1)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a* Φ_1)
then show *?case* **by** *auto*
qed

5.3 Interpretation of configurations

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

lemma *HeronConf-interp-composition*:
shows $\langle [\![\ \Gamma_1,\ n \vdash \Psi_1 \triangleright \Phi_1\ ]\!]_{config} \cap [\![\ \Gamma_2,\ n \vdash \Psi_2 \triangleright \Phi_2\ ]\!]_{config}$
$= [\![\ (\Gamma_1\ @\ \Gamma_2),\ n \vdash (\Psi_1\ @\ \Psi_2) \triangleright (\Phi_1\ @\ \Phi_2)\ ]\!]_{config}\rangle$
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 $\langle [\![\ \Gamma,\ n \vdash [] \triangleright \Phi\ ]\!]_{config}$
$= [\![\ \Gamma,\ \mathit{Suc}\ n \vdash \Phi \triangleright []\ ]\!]_{config}\rangle$
proof −
have $\langle [\![\ \Gamma,\ n \vdash [] \triangleright \Phi\ ]\!]_{config} = [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ []\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ \mathit{Suc}\ n}\rangle$
by *simp*
moreover have $\langle [\![\ \Gamma,\ \mathit{Suc}\ n \vdash \Phi \triangleright []\ ]\!]_{config} = [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ \mathit{Suc}\ n} \cap [\![[\ []\ ]\!]]_{TESL}{}^{\geq\ \mathit{Suc}\ n}\rangle$
by *simp*
moreover have $\langle [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ []\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ \mathit{Suc}\ n}$
$= [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ \mathit{Suc}\ n} \cap [\![[\ []\ ]\!]]_{TESL}{}^{\geq\ \mathit{Suc}\ n}\rangle$
by *simp*
ultimately show *?thesis* **by** *blast*
qed

lemma *HeronConf-interp-stepwise-sporadicon-cases*:
shows $\langle [\![\ \Gamma,\ n \vdash ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
$= [\![\ \Gamma,\ n \vdash \Psi \triangleright ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$\cup [\![\ ((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config}\rangle$
proof −
have $\langle [\![\ \Gamma,\ n \vdash (K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Psi \triangleright \Phi\ ]\!]_{config} = [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ (K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ \mathit{Suc}\ n}\rangle$
by *simp*
moreover have $\langle [\![\ \Gamma,\ n \vdash \Psi \triangleright ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Phi)\ ]\!]_{config} = [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ (K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Phi\ ]\!]]_{TESL}{}^{\geq\ \mathit{Suc}\ n}\rangle$
by *simp*
moreover have $\langle [\![\ ((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config} = [\![[\ ((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma)\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ \mathit{Suc}\ n}\rangle$
by *simp*
ultimately show *?thesis*
proof −
have $\langle ([\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2 \Downarrow n\ @\ \tau\ ]\!]_{prim} \cup [\![\ K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2\ ]\!]_{TESL}{}^{\geq\ \mathit{Suc}\ n}) \cap ([\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n}) = [\![\ K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2\ ]\!]_{TESL}{}^{\geq\ n} \cap ([\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ \Gamma\ ]\!]]_{prim})\rangle$
using *TESL-interp-stepwise-sporadicon-coind-unfold* **by** *blast*
then have $\langle [\![[\ ((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma)\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cup [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![\ K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2\ ]\!]_{TESL}{}^{\geq\ \mathit{Suc}\ n} = [\![[\ (K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ \Gamma\ ]\!]]_{prim}\rangle$
by *auto*

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

lemma *HeronConf-interp-stepwise-tagrel-cases*:
shows ‹$[\![\ \Gamma,\ n \vdash ((\mathit{time-relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
$= [\![\ ((\lfloor \tau_{var}(K_1,\ n),\ \tau_{var}(K_2,\ n) \rfloor \in R)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((\mathit{time-relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Phi)\ ]\!]_{config}$›
proof −
have ‹$[\![\ \Gamma,\ n \vdash (\mathit{time-relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Psi \triangleright \Phi\ ]\!]_{config} = [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ (\mathit{time-relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ Suc\ n}$›
by *simp*
moreover have ‹$[\![\ ((\lfloor \tau_{var}(K_1,\ n),\ \tau_{var}(K_2,\ n) \rfloor \in R)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((\mathit{time-relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Phi)\ ]\!]_{config}$
$= [\![[\ (\lfloor \tau_{var}(K_1,\ n),\ \tau_{var}(K_2,\ n) \rfloor \in R)\ \#\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ (\mathit{time-relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Phi\ ]\!]]_{TESL}{}^{\geq\ Suc\ n}$›
by *simp*
ultimately show *?thesis*
proof −
have ‹$[\![\ \lfloor \tau_{var}(K_1,\ n),\ \tau_{var}(K_2,\ n) \rfloor \in R\ ]\!]_{prim} \cap [\![\ \mathit{time-relation}\ \lfloor K_1,\ K_2 \rfloor \in R\ ]\!]_{TESL}{}^{\geq\ Suc\ n} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} = [\![[\ (\mathit{time-relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n}$›
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 ‹$[\![\ \Gamma,\ n \vdash ((K_1\ \mathit{implies}\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
$= [\![\ ((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$\cup [\![\ ((K_1 \Uparrow n)\ \#\ (K_2 \Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$›
proof −
have ‹$[\![\ \Gamma,\ n \vdash (K_1\ \mathit{implies}\ K_2)\ \#\ \Psi \triangleright \Phi\ ]\!]_{config} = [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ (K_1\ \mathit{implies}\ K_2)\ \#\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ Suc\ n}$›
by *simp*
moreover have ‹$[\![\ ((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2)\ \#\ \Phi)\ ]\!]_{config} = [\![[\ (K_1 \neg\Uparrow n)\ \#\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ (K_1\ \mathit{implies}\ K_2)\ \#\ \Phi\ ]\!]]_{TESL}{}^{\geq\ Suc\ n}$›
by *simp*
moreover have ‹$[\![\ ((K_1 \Uparrow n)\ \#\ (K_2 \Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2)\ \#\ \Phi)\ ]\!]_{config} = [\![[\ ((K_1 \Uparrow n)\ \#\ (K_2 \Uparrow n)\ \#\ \Gamma)\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ (K_1\ \mathit{implies}\ K_2)\ \#\ \Phi\ ]\!]]_{TESL}{}^{\geq\ Suc\ n}$›
by *simp*
ultimately show *?thesis*
proof −

have *f1*: ‹($[\![K_1 \neg\Uparrow n]\!]_{prim} \cup [\![K_1 \Uparrow n]\!]_{prim} \cap [\![K_2 \Uparrow n]\!]_{prim}) \cap [\![K_1$ *implies* $K_2]\!]_{TESL}{}^{\geq\, Suc\ n} \cap ([\![[\Psi]\!]]_{TESL}{}^{\geq\, n} \cap [\![[\Phi]\!]]_{TESL}{}^{\geq\, Suc\ n}) = [\![[(K_1$ *implies* $K_2) \# \Psi]\!]]_{TESL}{}^{\geq\, n} \cap [\![[\Phi]\!]]_{TESL}{}^{\geq\, Suc\ n}$›
using *TESL-interp-stepwise-implies-coind-unfold TESL-interpretation-stepwise-cons-morph* **by** *blast*
have ‹$[\![K_1 \neg\Uparrow n]\!]_{prim} \cap [\![[\Gamma]\!]]_{prim} \cup [\![K_1 \Uparrow n]\!]_{prim} \cap [\![[(K_2 \Uparrow n) \# \Gamma]\!]]_{prim} = ([\![K_1 \neg\Uparrow n]\!]_{prim} \cup [\![K_1 \Uparrow n]\!]_{prim} \cap [\![K_2 \Uparrow n]\!]_{prim}) \cap [\![[\Gamma]\!]]_{prim}$›
by *force*
then have ‹$[\![\Gamma, n \vdash ((K_1$ *implies* $K_2) \# \Psi) \triangleright \Phi]\!]_{config} = ([\![K_1 \neg\Uparrow n]\!]_{prim} \cap [\![[\Gamma]\!]]_{prim} \cup [\![K_1 \Uparrow n]\!]_{prim} \cap [\![[(K_2 \Uparrow n) \# \Gamma]\!]]_{prim}) \cap ([\![[\Psi]\!]]_{TESL}{}^{\geq\, n} \cap [\![[(K_1$ *implies* $K_2) \# \Phi]\!]]_{TESL}{}^{\geq\, Suc\ n})$›
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 ‹$[\![\Gamma, n \vdash ((K_1$ *implies not* $K_2) \# \Psi) \triangleright \Phi]\!]_{config}$
$= [\![((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1$ *implies not* $K_2) \# \Phi)]\!]_{config}$
$\cup [\![((K_1 \Uparrow n) \# (K_2 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1$ *implies not* $K_2) \# \Phi)]\!]_{config}$›
proof –
have ‹$[\![\Gamma, n \vdash (K_1$ *implies not* $K_2) \# \Psi \triangleright \Phi]\!]_{config} = [\![[\Gamma]\!]]_{prim} \cap [\![[(K_1$ *implies not* $K_2) \# \Psi]\!]]_{TESL}{}^{\geq\, n} \cap [\![[\Phi]\!]]_{TESL}{}^{\geq\, Suc\ n}$›
by *simp*
moreover have ‹$[\![((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1$ *implies not* $K_2) \# \Phi)]\!]_{config}$
$= [\![[(K_1 \neg\Uparrow n) \# \Gamma]\!]]_{prim} \cap [\![[\Psi]\!]]_{TESL}{}^{\geq\, n} \cap [\![[(K_1$ *implies not* $K_2) \# \Phi]\!]]_{TESL}{}^{\geq\, Suc\ n}$›
by *simp*
moreover have ‹$[\![((K_1 \Uparrow n) \# (K_2 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1$ *implies not* $K_2) \# \Phi)]\!]_{config}$
$= [\![[((K_1 \Uparrow n) \# (K_2 \neg\Uparrow n) \# \Gamma)]\!]]_{prim} \cap [\![[\Psi]\!]]_{TESL}{}^{\geq\, n} \cap [\![[(K_1$ *implies not* $K_2) \# \Phi]\!]]_{TESL}{}^{\geq\, Suc\ n}$›
by *simp*
ultimately show *?thesis*
proof –
have *f1*: ‹($[\![K_1 \neg\Uparrow n]\!]_{prim} \cup [\![K_1 \Uparrow n]\!]_{prim} \cap [\![K_2 \neg\Uparrow n]\!]_{prim}) \cap [\![K_1$ *implies not* $K_2]\!]_{TESL}{}^{\geq\, Suc\ n} \cap ([\![[\Psi]\!]]_{TESL}{}^{\geq\, n} \cap [\![[\Phi]\!]]_{TESL}{}^{\geq\, Suc\ n})$
$= [\![[(K_1$ *implies not* $K_2) \# \Psi]\!]]_{TESL}{}^{\geq\, n} \cap [\![[\Phi]\!]]_{TESL}{}^{\geq\, Suc\ n}$›
using *TESL-interp-stepwise-implies-not-coind-unfold TESL-interpretation-stepwise-cons-morph* **by** *blast*
have ‹$[\![K_1 \neg\Uparrow n]\!]_{prim} \cap [\![[\Gamma]\!]]_{prim} \cup [\![K_1 \Uparrow n]\!]_{prim} \cap [\![[(K_2 \neg\Uparrow n) \# \Gamma]\!]]_{prim} = ([\![K_1 \neg\Uparrow n]\!]_{prim} \cup [\![K_1 \Uparrow n]\!]_{prim} \cap [\![K_2 \neg\Uparrow n]\!]_{prim}) \cap [\![[\Gamma]\!]]_{prim}$›
by *force*
then have ‹$[\![\Gamma, n \vdash ((K_1$ *implies not* $K_2) \# \Psi) \triangleright \Phi]\!]_{config}$
$= ([\![K_1 \neg\Uparrow n]\!]_{prim} \cap [\![[\Gamma]\!]]_{prim} \cup [\![K_1 \Uparrow n]\!]_{prim} \cap [\![[(K_2 \neg\Uparrow n)$

$\#\ \Gamma\ ]\!]_{prim}) \cap ([\![\ \Psi\ ]\!]_{TESL}{}^{\geq\ n} \cap [\![\ (K_1\ \textit{implies not}\ K_2)\ \#\ \Phi\ ]\!]_{TESL}{}^{\geq\ Suc\ n})\rangle$
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 $\langle[\![\ \Gamma,\ n \vdash ((K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
$= [\![\ ((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Phi)\ ]\!]_{config}$
$\cup\ [\![\ ((K_1 \Uparrow n)\ \#\ (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Phi)\ ]\!]_{config}\rangle$
proof −
have *1*:$\langle[\![\ \Gamma,\ n \vdash (K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Psi \triangleright \Phi\ ]\!]_{config}$
$= [\![\![\ \Gamma\ ]\!]\!]_{prim} \cap [\![\![\ (K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Psi\ ]\!]\!]_{TESL}{}^{\geq\ n} \cap [\![\![\ \Phi\ ]\!]\!]_{TESL}{}^{\geq\ Suc\ n}\rangle$
by *simp*
moreover have $\langle[\![\ ((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Phi)\ ]\!]_{config}$
$= [\![\![\ (K_1 \neg\Uparrow n)\ \#\ \Gamma\ ]\!]\!]_{prim} \cap [\![\![\ \Psi\ ]\!]\!]_{TESL}{}^{\geq\ n} \cap [\![\![\ (K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Phi\ ]\!]\!]_{TESL}{}^{\geq\ Suc\ n}\rangle$
by *simp*
moreover have $\langle[\![\ ((K_1 \Uparrow n)\ \#\ (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Phi)\ ]\!]_{config}$
$= [\![\![\ (K_1 \Uparrow n)\ \#\ (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)\ \#\ \Gamma\ ]\!]\!]_{prim} \cap [\![\![\ \Psi\ ]\!]\!]_{TESL}{}^{\geq\ n}$
$\cap\ [\![\![\ (K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Phi\ ]\!]\!]_{TESL}{}^{\geq\ Suc\ n}\rangle$
by *simp*
ultimately show *?thesis*
proof −
have $\langle[\![\ \Gamma,\ n \vdash (K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Psi \triangleright \Phi\ ]\!]_{config} = [\![\![\ \Gamma\ ]\!]\!]_{prim} \cap ([\![\![\ (K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Psi\ ]\!]\!]_{TESL}{}^{\geq\ n} \cap [\![\![\ \Phi\ ]\!]\!]_{TESL}{}^{\geq\ Suc\ n})\rangle$
using *1* **by** *blast*
then have $\langle[\![\ \Gamma,\ n \vdash (K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Psi \triangleright \Phi\ ]\!]_{config} = ([\![\ K_1 \neg\Uparrow n\ ]\!]_{prim} \cup [\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2\ @\ n \oplus \delta\tau \Rightarrow K_3\ ]\!]_{prim}) \cap ([\![\![\ \Gamma\ ]\!]\!]_{prim} \cap ([\![\![\ \Psi\ ]\!]\!]_{TESL}{}^{\geq\ n} \cap [\![\![\ (K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Phi\ ]\!]\!]_{TESL}{}^{\geq\ Suc\ n}))\rangle$
using *TESL-interpretation-stepwise-cons-morph TESL-interp-stepwise-timedelayed-coind-unfold*
proof −
have $\langle[\![\![\ (K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Psi\ ]\!]\!]_{TESL}{}^{\geq\ n} = ([\![\ K_1 \neg\Uparrow n\ ]\!]_{prim} \cup [\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2\ @\ n \oplus \delta\tau \Rightarrow K_3\ ]\!]_{prim}) \cap [\![\ K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3\ ]\!]_{TESL}{}^{\geq\ Suc\ n} \cap [\![\![\ \Psi\ ]\!]\!]_{TESL}{}^{\geq\ n}\rangle$
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 ‹$[\![\ \Gamma,\ n \vdash ((K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
$= [\![\ ((\lceil \#^{\leq}\ K_2\ n,\ \#^{\leq}\ K_1\ n \rceil \in (\lambda(x,y).\ x \leq y))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$›
proof −
have ‹$[\![\ \Gamma,\ n \vdash (K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Psi \triangleright \Phi\ ]\!]_{config} = [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ (K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Psi\ ]\!]]_{TESL}^{\geq\ n} \cap [\![[\ \Phi\ ]\!]]_{TESL}^{\geq\ \textit{Suc}\ n}$›
by *simp*
moreover have ‹$[\![\ ((\lceil \#^{\leq}\ K_2\ n,\ \#^{\leq}\ K_1\ n \rceil \in (\lambda(x,y).\ x \leq y))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$= [\![[\ (\lceil \#^{\leq}\ K_2\ n,\ \#^{\leq}\ K_1\ n \rceil \in (\lambda(x,y).\ x \leq y))\ \#\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}^{\geq\ n} \cap [\![[\ (K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Phi\ ]\!]]_{TESL}^{\geq\ \textit{Suc}\ n}$›
by *simp*
ultimately show *?thesis*
proof −
have ‹$[\![\ \lceil \#^{\leq}\ K_2\ n,\ \#^{\leq}\ K_1\ n \rceil \in (\lambda(x,y).\ x \leq y)\ ]\!]_{prim} \cap [\![\ K_1\ \textit{weakly precedes}\ K_2\ ]\!]_{TESL}^{\geq\ \textit{Suc}\ n} \cap [\![[\ \Psi\ ]\!]]_{TESL}^{\geq\ n} = [\![[\ (K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Psi\ ]\!]]_{TESL}^{\geq\ n}$›
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-weakly-precedes-cases'*:
shows ‹$[\![\ \Gamma,\ n \vdash ((K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
$= [\![\ (((\#^{\leq}\ K_2\ n) \preceq (\#^{\leq}\ K_1\ n))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$›
oops

lemma *HeronConf-interp-stepwise-strictly-precedes-cases*:
shows ‹$[\![\ \Gamma,\ n \vdash ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
$= [\![\ ((\lceil \#^{\leq}\ K_2\ n,\ \#^{<}\ K_1\ n \rceil \in (\lambda(x,y).\ x \leq y))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$›
proof −
have ‹$[\![\ \Gamma,\ n \vdash (K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi \triangleright \Phi\ ]\!]_{config} = [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ (K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi\ ]\!]]_{TESL}^{\geq\ n} \cap [\![[\ \Phi\ ]\!]]_{TESL}^{\geq\ \textit{Suc}\ n}$›
by *simp*
moreover have ‹$[\![\ ((\lceil \#^{\leq}\ K_2\ n,\ \#^{<}\ K_1\ n \rceil \in (\lambda(x,y).\ x \leq y))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$= [\![[\ (\lceil \#^{\leq}\ K_2\ n,\ \#^{<}\ K_1\ n \rceil \in (\lambda(x,y).\ x \leq y))\ \#\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}^{\geq\ n} \cap [\![[\ (K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi\ ]\!]]_{TESL}^{\geq\ \textit{Suc}\ n}$›
by *simp*
ultimately show *?thesis*

proof –
have $\langle [\![\lceil \#^{\leq} K_2\ n,\ \#^{<} K_1\ n \rceil \in (\lambda(x,y).\ x \leq y)]\!]_{prim} \cap [\![K_1$ *strictly precedes* $K_2]\!]_{TESL}{}^{\geq\ Suc\ n} \cap [\![[\![\Psi]\!]]\!]_{TESL}{}^{\geq\ n} = [\![[\![(K_1$ *strictly precedes* $K_2) \# \Psi]\!]]\!]_{TESL}{}^{\geq\ n}\rangle$
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 $\langle [\![\Gamma,\ n \vdash ((K_1$ *kills* $K_2) \# \Psi) \triangleright \Phi]\!]_{config}$
$= [\![((K_1 \neg\Uparrow n) \# \Gamma),\ n \vdash \Psi \triangleright ((K_1$ *kills* $K_2) \# \Phi)]\!]_{config}$
$\cup [\![((K_1 \Uparrow n) \# (K_2 \neg\Uparrow \geq n) \# \Gamma),\ n \vdash \Psi \triangleright ((K_1$ *kills* $K_2) \# \Phi)]\!]_{config}\rangle$
proof –
have $\langle [\![\Gamma,\ n \vdash ((K_1$ *kills* $K_2) \# \Psi) \triangleright \Phi]\!]_{config} = [\![[\![\Gamma]\!]]\!]_{prim} \cap [\![[\![(K_1$ *kills* $K_2) \# \Psi]\!]]\!]_{TESL}{}^{\geq\ n} \cap [\![[\![\Phi]\!]]\!]_{TESL}{}^{\geq\ Suc\ n}\rangle$
by *simp*
moreover have $\langle [\![((K_1 \neg\Uparrow n) \# \Gamma),\ n \vdash \Psi \triangleright ((K_1$ *kills* $K_2) \# \Phi)]\!]_{config}$
$= [\![[\![(K_1 \neg\Uparrow n) \# \Gamma]\!]]\!]_{prim} \cap [\![[\![\Psi]\!]]\!]_{TESL}{}^{\geq\ n} \cap [\![[\![(K_1$ *kills* $K_2) \# \Phi]\!]]\!]_{TESL}{}^{\geq\ Suc\ n}\rangle$
by *simp*
moreover have $\langle [\![((K_1 \Uparrow n) \# (K_2 \neg\Uparrow \geq n) \# \Gamma),\ n \vdash \Psi \triangleright ((K_1$ *kills* $K_2) \# \Phi)]\!]_{config}$
$= [\![[\![(K_1 \Uparrow n) \# (K_2 \neg\Uparrow \geq n) \# \Gamma]\!]]\!]_{prim} \cap [\![[\![\Psi]\!]]\!]_{TESL}{}^{\geq\ n} \cap [\![[\![(K_1$ *kills* $K_2) \# \Phi]\!]]\!]_{TESL}{}^{\geq\ Suc\ n}\rangle$
by *simp*
ultimately show *?thesis*
proof –
have $\langle [\![[\![(K_1$ *kills* $K_2) \# \Psi]\!]]\!]_{TESL}{}^{\geq\ n} = ([\![(K_1 \neg\Uparrow n)]\!]_{prim} \cup [\![(K_1 \Uparrow n)]\!]_{prim} \cap [\![(K_2 \neg\Uparrow \geq n)]\!]_{prim}) \cap [\![(K_1$ *kills* $K_2)]\!]_{TESL}{}^{\geq\ Suc\ n} \cap [\![[\![\Psi]\!]]\!]_{TESL}{}^{\geq\ n}\rangle$
using *TESL-interp-stepwise-kills-coind-unfold TESL-interpretation-stepwise-cons-morph*
by *blast*
then show *?thesis*
by *auto*
qed
qed

end

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 $[], \mathit{0} \vdash \Psi \triangleright []$

theorem *solve-start*:
shows $\langle [\![[\![\ \Psi \ ]\!]]\!]_{TESL} = [\![\ [], \mathit{0} \vdash \Psi \triangleright [] \ ]\!]_{config} \rangle$
proof –
have $\langle [\![[\![\ \Psi \ ]\!]]\!]_{TESL} = [\![[\![\ \Psi \ ]\!]]\!]_{TESL}{}^{\geq \ \mathit{0}} \rangle$
by (*simp add*: *TESL-interpretation-stepwise-zero'*)
moreover have $\langle [\![\ [], \mathit{0} \vdash \Psi \triangleright [] \ ]\!]_{config} = [\![[\![\ [] \ ]\!]]\!]_{prim} \cap [\![[\![\ \Psi \ ]\!]]\!]_{TESL}{}^{\geq \ \mathit{0}} \cap [\![[\![\ [] \ ]\!]]\!]_{TESL}{}^{\geq \ \mathit{Suc} \ \mathit{0}} \rangle$
by *simp*
ultimately show *?thesis* **by** *auto*
qed

6.2 Soundness

lemma *sound-reduction*:
assumes $\langle (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) \rangle$
shows $\langle [\![[\![\ \Gamma_1 \ ]\!]]\!]_{prim} \cap [\![[\![\ \Psi_1 \ ]\!]]\!]_{TESL}{}^{\geq \ n_1} \cap [\![[\![\ \Phi_1 \ ]\!]]\!]_{TESL}{}^{\geq \ \mathit{Suc} \ n_1}$
$\supseteq [\![[\![\ \Gamma_2 \ ]\!]]\!]_{prim} \cap [\![[\![\ \Psi_2 \ ]\!]]\!]_{TESL}{}^{\geq \ n_2} \cap [\![[\![\ \Phi_2 \ ]\!]]\!]_{TESL}{}^{\geq \ \mathit{Suc} \ n_2} \rangle$ (**is** *?P*)
proof –
from *assms* **consider**
(*a*) $\langle (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow_i (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) \rangle$
| (*b*) $\langle (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow_e (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) \rangle$
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 ‹$(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash (K_1\ \textit{sporadic}\ \tau\ \textit{on}\ K_2)\ \#\ \Psi \triangleright \Phi)$›
and ‹$(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (\Gamma,\ n \vdash \Psi \triangleright ((K_1\ \textit{sporadic}\ \tau\ \textit{on}\ K_2)\ \#\ \Phi))$›
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 ‹$(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash (K_1\ \textit{sporadic}\ \tau\ \textit{on}\ K_2)\ \#\ \Psi \triangleright \Phi)$›
and ‹$(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi)$›
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 ‹$(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash (\textit{time}-\textit{relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Psi \triangleright \Phi)$›
and ‹$(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((\lfloor \tau_{var}\ (K_1,\ n),\ \tau_{var}\ (K_2,\ n) \rfloor \in R)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((\textit{time}-\textit{relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Phi))$›
thus *?P*
using *HeronConf-interp-stepwise-tagrel-cases HeronConf-interpretation.simps* **by** *blast*
next
fix $\Gamma\ n\ K_1\ K_2\ \Psi\ \Phi$
assume ‹$(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash (K_1\ \textit{implies}\ K_2)\ \#\ \Psi \triangleright \Phi)$›
and ‹$(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{implies}\ K_2)\ \#\ \Phi))$›
thus *?P*
using *HeronConf-interp-stepwise-implies-cases HeronConf-interpretation.simps* **by** *blast*
next
fix $\Gamma\ n\ K_1\ K_2\ \Psi\ \Phi$
assume ‹$(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash ((K_1\ \textit{implies}\ K_2)\ \#\ \Psi) \triangleright \Phi)$›
and ‹$(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((K_1 \Uparrow n)\ \#\ (K_2 \Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{implies}\ K_2)\ \#\ \Phi))$›
thus *?P*
using *HeronConf-interp-stepwise-implies-cases HeronConf-interpretation.simps* **by** *blast*
next
fix $\Gamma\ n\ K_1\ K_2\ \Psi\ \Phi$
assume ‹$(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash ((K_1\ \textit{implies not}\ K_2)\ \#\ \Psi) \triangleright \Phi)$›
and ‹$(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{implies not}\ K_2)$

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

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

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

lemma *sound-reduction′*:
assumes ‹$\mathcal{S}_1 \hookrightarrow \mathcal{S}_2$›
shows ‹$[\![\ \mathcal{S}_1\ ]\!]_{config} \supseteq [\![\ \mathcal{S}_2\ ]\!]_{config}$›
proof −
have ‹$\forall\, s_1\ s_2.\ ([\![\ s_2\ ]\!]_{config} \subseteq [\![\ s_1\ ]\!]_{config}) \vee \neg(s_1 \hookrightarrow s_2)$›
using *sound-reduction* **by** *fastforce*
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *sound-reduction-generalized*:
assumes ‹$\mathcal{S}_1 \hookrightarrow^{k} \mathcal{S}_2$›
shows ‹$[\![\ \mathcal{S}_1\ ]\!]_{config} \supseteq [\![\ \mathcal{S}_2\ ]\!]_{config}$›
proof −
from *assms* **show** *?thesis*
proof (*induct k arbitrary*: $\mathcal{S}_2$)
case *0*
hence $*$: ‹$\mathcal{S}_1 \hookrightarrow^{0} \mathcal{S}_2 \Longrightarrow \mathcal{S}_1 = \mathcal{S}_2$› **by** *auto*
moreover have ‹$\mathcal{S}_1 = \mathcal{S}_2$› **using** $*$ *0.prems* **by** *linarith*
ultimately show *?case* **by** *auto*
next
case (*Suc k*)
thus *?case*
proof −
fix k :: *nat*
assume *ff*: ‹$\mathcal{S}_1 \hookrightarrow^{Suc\ k} \mathcal{S}_2$›
assume *hi*: ‹$\bigwedge\mathcal{S}_2.\ \mathcal{S}_1 \hookrightarrow^{k} \mathcal{S}_2 \Longrightarrow [\![\ \mathcal{S}_2\ ]\!]_{config} \subseteq [\![\ \mathcal{S}_1\ ]\!]_{config}$›
obtain $\mathcal{S}_n$ **where** *red-decomp*: ‹$(\mathcal{S}_1 \hookrightarrow^{k} \mathcal{S}_n) \wedge (\mathcal{S}_n \hookrightarrow \mathcal{S}_2)$› **using** *ff* **by** *auto*
hence ‹$[\![\ \mathcal{S}_1\ ]\!]_{config} \supseteq [\![\ \mathcal{S}_n\ ]\!]_{config}$› **using** *hi* **by** *simp*
also have ‹$[\![\ \mathcal{S}_n\ ]\!]_{config} \supseteq [\![\ \mathcal{S}_2\ ]\!]_{config}$› **by** (*simp add*: *red-decomp sound-reduction′*)

ultimately show ‹$[\![\ \mathcal{S}_1\ ]\!]_{config} \supseteq [\![\ \mathcal{S}_2\ ]\!]_{config}$› **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 ‹$([],\ 0 \vdash \Psi \triangleright []) \hookrightarrow^k \mathcal{S}$›
shows ‹$[\![[\![\ \Psi\ ]\!]]\!]_{TESL} \supseteq [\![\ \mathcal{S}\ ]\!]_{config}$›
using *assms sound-reduction-generalized solve-start* **by** *blast*

6.3 Completeness

lemma *complete-direct-successors*:
shows ‹$[\![\ \Gamma,\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config} \subseteq (\bigcup X{\in}\mathcal{C}_{next}\ (\Gamma,\ n \vdash \Psi \triangleright \Phi).\ [\![\ X\ ]\!]_{config})$›
proof (*induct* Ψ)
case *Nil*
show *?case*
using *HeronConf-interp-stepwise-instant-cases operational-semantics-step.simps operational-semantics-intro.instant-i*
by *fastforce*
next
case (*Cons* ψ Ψ)
then show *?case*
proof (*cases* ψ)
case (*SporadicOn K1* τ *K2*)
then show *?thesis*
using *HeronConf-interp-stepwise-sporadicon-cases*[*of* ‹Γ› ‹n› ‹*K1*› ‹τ› ‹*K2*› ‹Ψ› ‹Φ›]
Cnext-solve-sporadicon[*of* ‹Γ› ‹n› ‹Ψ› ‹*K1*› ‹τ› ‹*K2*› ‹Φ›] **by** *blast*
next
case (*TagRelation* K_1 K_2 R)
then show *?thesis*
using *HeronConf-interp-stepwise-tagrel-cases*[*of* ‹Γ› ‹n› ‹K_1› ‹K_2› ‹R› ‹Ψ› ‹Φ›]
Cnext-solve-tagrel[*of* ‹K_1› ‹n› ‹K_2› ‹R› ‹Γ› ‹Ψ› ‹Φ›] **by** *blast*
next
case (*Implies K1 K2*)
then show *?thesis*
using *HeronConf-interp-stepwise-implies-cases*[*of* ‹Γ› ‹n› ‹*K1*› ‹*K2*› ‹Ψ› ‹Φ›]
Cnext-solve-implies[*of* ‹*K1*› ‹n› ‹Γ› ‹Ψ› ‹*K2*› ‹Φ›] **by** *blast*
next
case (*ImpliesNot K1 K2*)
then show *?thesis*
using *HeronConf-interp-stepwise-implies-not-cases*[*of* ‹Γ› ‹n› ‹*K1*› ‹*K2*› ‹Ψ› ‹Φ›]
Cnext-solve-implies-not[*of* ‹*K1*› ‹n› ‹Γ› ‹Ψ› ‹*K2*› ‹Φ›] **by** *blast*

next
case (*TimeDelayedBy Kmast* τ *Kmeas Kslave*)
thus *?thesis*
using *HeronConf-interp-stepwise-timedelayed-cases*[*of* ‹Γ› ‹*n*› ‹*Kmast*› ‹τ› ‹*Kmeas*› ‹*Kslave*› ‹Ψ› ‹Φ›]
Cnext-solve-timedelayed[*of* ‹*Kmast*› ‹*n*› ‹Γ› ‹Ψ› ‹τ› ‹*Kmeas*› ‹*Kslave*› ‹Φ›] **by** *blast*
next
case (*WeaklyPrecedes K1 K2*)
then show *?thesis*
using *HeronConf-interp-stepwise-weakly-precedes-cases*[*of* ‹Γ› ‹*n*› ‹*K1*› ‹*K2*› ‹Ψ› ‹Φ›]
Cnext-solve-weakly-precedes[*of* ‹*K2*› ‹*n*› ‹*K1*› ‹Γ› ‹Ψ› ‹Φ›]
by *blast*
next
case (*StrictlyPrecedes K1 K2*)
then show *?thesis*
using *HeronConf-interp-stepwise-strictly-precedes-cases*[*of* ‹Γ› ‹*n*› ‹*K1*› ‹*K2*› ‹Ψ› ‹Φ›]
Cnext-solve-strictly-precedes[*of* ‹*K2*› ‹*n*› ‹*K1*› ‹Γ› ‹Ψ› ‹Φ›]
by *blast*
next
case (*Kills K1 K2*)
then show *?thesis*
using *HeronConf-interp-stepwise-kills-cases*[*of* ‹Γ› ‹*n*› ‹*K1*› ‹*K2*› ‹Ψ› ‹Φ›]
Cnext-solve-kills[*of* ‹*K1*› ‹*n*› ‹Γ› ‹Ψ› ‹*K2*› ‹Φ›] **by** *blast*
qed
qed

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

lemma *branch-existence*:
assumes ‹$\varrho \in [\![\ \mathcal{S}_1\ ]\!]_{config}$›
shows ‹$\exists \mathcal{S}_2.\ (\mathcal{S}_1 \hookrightarrow \mathcal{S}_2) \wedge (\varrho \in [\![\ \mathcal{S}_2\ ]\!]_{config})$›
proof −
from *assms complete-direct-successors'* **have** ‹$\varrho \in (\bigcup X{\in}\mathcal{C}_{next}\ \mathcal{S}_1.\ [\![\ X\ ]\!]_{config})$› **by** *blast*
hence ‹$\exists\, s{\in}\mathcal{C}_{next}\ \mathcal{S}_1.\ \varrho \in [\![\ s\ ]\!]_{config}$› **by** *simp*
thus *?thesis* **by** *blast*
qed

lemma *branch-existence'*:
assumes ‹$\varrho \in [\![\ \mathcal{S}_1\ ]\!]_{config}$›

shows ‹$\exists\mathcal{S}_2.\ (\mathcal{S}_1 \hookrightarrow^{k} \mathcal{S}_2) \wedge (\varrho \in [\![\ \mathcal{S}_2\ ]\!]_{config})$›
proof (*induct k*)
case *0*
then show *?case* **by** (*simp add*: *assms*)
next
case (*Suc k*)
then show *?case*
using *branch-existence relpowp-Suc-I*[*of* ‹*k*› ‹*operational-semantics-step*›] **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 ‹$\varrho \in [\![[\![\ \Psi\ ]\!]]\!]_{TESL}$›
shows ‹$\exists\mathcal{S}.\ (([],\ 0 \vdash \Psi \triangleright [])\ \hookrightarrow^{k}\ \mathcal{S})$
$\wedge\ \varrho \in [\![\ \mathcal{S}\ ]\!]_{config}$›
using *assms branch-existence′ solve-start* **by** *blast*

6.4 Progress

lemma *instant-index-increase*:
assumes ‹$\varrho \in [\![\ \Gamma,\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config}$›
shows ‹$\exists\Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((\Gamma,\ n \vdash \Psi \triangleright \Phi)\ \hookrightarrow^{k}\ (\Gamma_k,\ Suc\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\ \Gamma_k,\ Suc\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}$›
proof (*insert assms*, *induct* Ψ *arbitrary*: $\Gamma\ \Phi$)
case (*Nil* $\Gamma\ \Phi$)
then show *?case*
proof −
have ‹$(\Gamma,\ n \vdash [] \triangleright \Phi) \hookrightarrow^{1} (\Gamma,\ Suc\ n \vdash \Phi \triangleright [])$›
using *instant-i intro-part* **by** *fastforce*
moreover have ‹$[\![\ \Gamma,\ n \vdash [] \triangleright \Phi\ ]\!]_{config} = [\![\ \Gamma,\ Suc\ n \vdash \Phi \triangleright []\ ]\!]_{config}$›
by *auto*
moreover have ‹$\varrho \in [\![\ \Gamma,\ Suc\ n \vdash \Phi \triangleright []\ ]\!]_{config}$›
using *assms Nil.prems calculation*(*2*) **by** *blast*
ultimately show *?thesis* **by** *blast*
qed
next
case (*Cons* $\psi\ \Psi$)
then show *?case*
proof (*induct* ψ)
case (*SporadicOn* $K_1\ \tau\ K_2$)
have *branches*: ‹$[\![\ \Gamma,\ n \vdash ((K_1\ \textit{sporadic}\ \tau\ \textit{on}\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
$= [\![\ \Gamma,\ n \vdash \Psi \triangleright ((K_1\ \textit{sporadic}\ \tau\ \textit{on}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$\cup\ [\![\ ((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config}$›
using *HeronConf-interp-stepwise-sporadicon-cases* **by** *simp*
have *br1*: ‹$\varrho \in [\![\ \Gamma,\ n \vdash \Psi \triangleright ((K_1\ \textit{sporadic}\ \tau\ \textit{on}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$\Longrightarrow \exists\Gamma_k\ \Psi_k\ \Phi_k\ k.$

$((\Gamma, n \vdash ((K_1 \textit{ sporadic } \tau \textit{ on } K_2) \# \Psi) \triangleright \Phi) \hookrightarrow^k (\Gamma_k, \textit{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\Gamma_k, \textit{Suc } n \vdash \Psi_k \triangleright \Phi_k]\!]_{config}$›
proof −
assume *h1*: ‹$\varrho \in [\![\Gamma, n \vdash \Psi \triangleright ((K_1 \textit{ sporadic } \tau \textit{ on } K_2) \# \Phi)]\!]_{config}$›
hence ‹$\exists \Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((\Gamma, n \vdash \Psi \triangleright ((K_1 \textit{ sporadic } \tau \textit{ on } K_2) \# \Phi))$
$\hookrightarrow^k (\Gamma_k, \textit{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ (\varrho \in [\![\Gamma_k, \textit{Suc } n \vdash \Psi_k \triangleright \Phi_k]\!]_{config})$›
using *h1 SporadicOn.prems* **by** *simp*
thus *?thesis* **by** (*meson elims-part relpowp-Suc-I2 sporadic-on-e1*)
qed
have *br2*: ‹$\varrho \in [\![((K_1 \Uparrow n) \# (K_2 \Downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi]\!]_{config}$
$\Longrightarrow \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((\Gamma, n \vdash ((K_1 \textit{ sporadic } \tau \textit{ on } K_2) \# \Psi) \triangleright \Phi)$
$\hookrightarrow^k (\Gamma_k, \textit{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\Gamma_k, \textit{Suc } n \vdash \Psi_k \triangleright \Phi_k]\!]_{config}$›
proof −
assume *h2*: ‹$\varrho \in [\![((K_1 \Uparrow n) \# (K_2 \Downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi]\!]_{config}$›
hence ‹$\exists \Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((((K_1 \Uparrow n) \# (K_2 \Downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi)$
$\hookrightarrow^k (\Gamma_k, \textit{Suc } n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\Gamma_k, \textit{Suc } n \vdash \Psi_k \triangleright \Phi_k]\!]_{config}$›
using *h2 SporadicOn.prems* **by** *simp*
thus *?thesis*
by (*meson elims-part relpowp-Suc-I2 sporadic-on-e2*)
qed
from *branches SporadicOn.prems*(*2*) **have**
‹$\varrho \in [\![\Gamma, n \vdash \Psi \triangleright ((K_1 \textit{ sporadic } \tau \textit{ on } K_2) \# \Phi)]\!]_{config}$
$\cup\ [\![((K_1 \Uparrow n) \# (K_2 \Downarrow n @ \tau) \# \Gamma), n \vdash \Psi \triangleright \Phi]\!]_{config}$›
by *simp*
with *br1 br2* **show** *?case* **by** *blast*
next
case (*TagRelation* K_1 K_2 *R*)
have *branches*: ‹$[\![\Gamma, n \vdash ((\textit{time-relation } \lfloor K_1, K_2 \rfloor \in R) \# \Psi) \triangleright \Phi]\!]_{config}$
$= [\![((\lfloor \tau_{var}(K_1, n), \tau_{var}(K_2, n) \rfloor \in R) \# \Gamma), n \vdash \Psi \triangleright ((\textit{time-relation } \lfloor K_1, K_2 \rfloor \in R) \# \Phi)]\!]_{config}$›
using *HeronConf-interp-stepwise-tagrel-cases* **by** *simp*
thus *?case*
proof −
have ‹$\exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$
$((((\lfloor \tau_{var}(K_1, n), \tau_{var}(K_2, n) \rfloor \in R) \# \Gamma), n \vdash \Psi \triangleright ((\textit{time-relation } \lfloor K_1, K_2 \rfloor \in R) \# \Phi))$
$\hookrightarrow^k (\Gamma_k, \textit{Suc } n \vdash \Psi_k \triangleright \Phi_k)) \wedge \varrho \in [\![\Gamma_k, \textit{Suc } n \vdash \Psi_k \triangleright \Phi_k]\!]_{config}$›
using *TagRelation.prems* **by** *simp*
thus *?thesis*
by (*meson elims-part relpowp-Suc-I2 tagrel-e*)
qed
next
case (*Implies* K_1 K_2)
have *branches*: ‹$[\![\Gamma, n \vdash ((K_1 \textit{ implies } K_2) \# \Psi) \triangleright \Phi]\!]_{config}$
$= [\![((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \textit{ implies } K_2) \# \Phi)]\!]_{config}$

$\cup$ $[\![$ $((K_1 \Uparrow n)$ # $(K_2 \Uparrow n)$ # $\Gamma)$, $n \vdash \Psi \triangleright ((K_1$ *implies* $K_2)$ # $\Phi)$ $]\!]_{config}$⟩
using *HeronConf-interp-stepwise-implies-cases* **by** *simp*
have *br1*: ⟨$\varrho \in [\![$ $((K_1 \neg\Uparrow n)$ # $\Gamma)$, $n \vdash \Psi \triangleright ((K_1$ *implies* $K_2)$ # $\Phi)$ $]\!]_{config}$
$\Longrightarrow \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$ $((\Gamma, n \vdash ((K_1$ *implies* $K_2)$ # $\Psi) \triangleright \Phi) \hookrightarrow^k (\Gamma_k,$ *Suc* $n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![$ Γ_k, *Suc* $n \vdash \Psi_k \triangleright \Phi_k$ $]\!]_{config}$⟩
proof –
assume *h1*: ⟨$\varrho \in [\![$ $((K_1 \neg\Uparrow n)$ # $\Gamma)$, $n \vdash \Psi \triangleright ((K_1$ *implies* $K_2)$ # $\Phi)$ $]\!]_{config}$⟩
then have ⟨$\exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$
$((((K_1 \neg\Uparrow n)$ # $\Gamma)$, $n \vdash \Psi \triangleright ((K_1$ *implies* $K_2)$ # $\Phi)) \hookrightarrow^k (\Gamma_k,$ *Suc* $n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![$ Γ_k, *Suc* $n \vdash \Psi_k \triangleright \Phi_k$ $]\!]_{config}$⟩
using *h1 Implies.prems* **by** *simp*
then show *?thesis*
by (*meson elims-part relpowp-Suc-I2 implies-e1*)
qed
moreover have *br2*: ⟨$\varrho \in [\![$ $((K_1 \Uparrow n)$ # $(K_2 \Uparrow n)$ # $\Gamma)$, $n \vdash \Psi \triangleright ((K_1$ *implies* $K_2)$ # $\Phi)$ $]\!]_{config}$
$\Longrightarrow \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$ $((\Gamma, n \vdash ((K_1$ *implies* $K_2)$ # $\Psi) \triangleright \Phi)$
$\hookrightarrow^k (\Gamma_k,$ *Suc* $n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![$ Γ_k, *Suc* $n \vdash \Psi_k \triangleright \Phi_k$ $]\!]_{config}$⟩
proof –
assume *h2*: ⟨$\varrho \in [\![$ $((K_1 \Uparrow n)$ # $(K_2 \Uparrow n)$ # $\Gamma)$, $n \vdash \Psi \triangleright ((K_1$ *implies* $K_2)$ # $\Phi)$ $]\!]_{config}$⟩
then have ⟨$\exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$ (
$(((K_1 \Uparrow n)$ # $(K_2 \Uparrow n)$ # $\Gamma)$, $n \vdash \Psi \triangleright ((K_1$ *implies* $K_2)$ # $\Phi)) \hookrightarrow^k (\Gamma_k,$ *Suc* $n \vdash \Psi_k \triangleright \Phi_k)$
) $\wedge\ \varrho \in [\![$ Γ_k, *Suc* $n \vdash \Psi_k \triangleright \Phi_k$ $]\!]_{config}$⟩
using *h2 Implies.prems* **by** *simp*
thus *?thesis*
by (*meson elims-part relpowp-Suc-I2 implies-e2*)
qed
ultimately show *?case*
using *Implies.prems*(*2*) **by** *fastforce*
next
case (*ImpliesNot* K_1 K_2)
thus *?case* **using** *HeronConf-interp-stepwise-implies-not-cases Un-iff elims-part implies-not-e1 implies-not-e2 relpowp-Suc-I2*
by (*metis* (*no-types*, *lifting*))
next
case (*TimeDelayedBy* K_1 $\delta\tau$ K_2 K_3)
have *branches*: ⟨$[\![$ Γ, $n \vdash ((K_1$ *time−delayed by* $\delta\tau$ *on* K_2 *implies* $K_3)$ # $\Psi)$ $\triangleright\ \Phi$ $]\!]_{config}$
$=$ $[\![$ $((K_1 \neg\Uparrow n)$ # $\Gamma)$, $n \vdash \Psi \triangleright ((K_1$ *time−delayed by* $\delta\tau$ *on* K_2 *implies* $K_3)$ # $\Phi)$ $]\!]_{config}$
$\cup$ $[\![$ $((K_1 \Uparrow n)$ # $(K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)$ # $\Gamma)$, $n \vdash \Psi \triangleright ((K_1$ *time−delayed by* $\delta\tau$ *on* K_2 *implies* $K_3)$ # $\Phi)$ $]\!]_{config}$⟩
using *HeronConf-interp-stepwise-timedelayed-cases* **by** *simp*

have *br1*: ‹$\varrho \in [\![\ ((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{time-delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3)\ \#\ \Phi)\ ]\!]_{config}$
$\Longrightarrow \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$
$((\Gamma,\ n \vdash ((K_1\ \mathit{time-delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3)\ \#\ \Psi) \triangleright \Phi) \hookrightarrow^k (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\ \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}$›
proof −
assume *h1*: ‹$\varrho \in [\![\ ((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{time-delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3)\ \#\ \Phi)\ ]\!]_{config}$›
then have ‹$\exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$
$((((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{time-delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3)\ \#\ \Phi)) \hookrightarrow^k (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\ \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}$›
using *h1 TimeDelayedBy.prems* **by** *simp*
then show *?thesis*
by (*meson elims-part relpowp-Suc-I2 timedelayed-e1*)
qed
moreover have *br2*: ‹$\varrho \in [\![\ ((K_1 \Uparrow n)\ \#\ (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{time-delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3)\ \#\ \Phi)\ ]\!]_{config}$
$\Longrightarrow \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$
$((\Gamma,\ n \vdash ((K_1\ \mathit{time-delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3)\ \#\ \Psi) \triangleright \Phi) \hookrightarrow^k (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\ \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}$›
proof −
assume *h2*: ‹$\varrho \in [\![\ ((K_1 \Uparrow n)\ \#\ (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{time-delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3)\ \#\ \Phi)\ ]\!]_{config}$›
then have ‹$\exists \Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((((K_1 \Uparrow n)\ \#\ (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{time-delayed\ by}\ \delta\tau\ \mathit{on}\ K_2\ \mathit{implies}\ K_3)\ \#\ \Phi)) \hookrightarrow^k (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))\ \wedge\ \varrho \in [\![\ \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}$›
using *h2 TimeDelayedBy.prems* **by** *simp*
then show *?thesis*
by (*meson elims-part relpowp-Suc-I2 timedelayed-e2 sporadic-on-e1*)
qed
ultimately show *?case*
using *TimeDelayedBy.prems*(*2*) *HeronConf-interp-stepwise-timedelayed-cases* **by** *blast*
next
case (*WeaklyPrecedes* K_1 K_2)
then show *?case*
by (*metis* (*no-types*, *lifting*) *HeronConf-interp-stepwise-weakly-precedes-cases elims-part*
weakly-precedes-e relpowp-Suc-I2)
next
case (*StrictlyPrecedes* K_1 K_2)
then show *?case*
by (*metis* (*no-types*, *lifting*) *HeronConf-interp-stepwise-strictly-precedes-cases elims-part*
strictly-precedes-e relpowp-Suc-I2)
next

case (*Kills* K_1 K_2)
then show *?case*
by (*metis* (*no-types*, *lifting*) *HeronConf-interp-stepwise-kills-cases UnE elims-part kills-e1 kills-e2 relpowp-Suc-I2*)
qed
qed

lemma *instant-index-increase-generalized*:
assumes ‹$n < n_k$›
assumes ‹$\varrho \in [\![\ \Gamma,\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config}$›
shows ‹$\exists\,\Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((\Gamma,\ n \vdash \Psi \triangleright \Phi) \hookrightarrow^{k} (\Gamma_k,\ n_k \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\ \Gamma_k,\ n_k \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}$›
proof −
obtain δk **where** *diff*: ‹$n_k = \delta k + \mathit{Suc}\ n$›
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*: ‹$\varrho \in [\![\ \Gamma,\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config} \Longrightarrow \exists\,\Gamma_k\ \Psi_k\ \Phi_k\ k.$
$((\Gamma,\ n \vdash \Psi \triangleright \Phi) \hookrightarrow^{k} (\Gamma_k,\ \delta k + \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\ \Gamma_k,\ \delta k + \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}$›
using *Suc.hyps* **by** *blast*
obtain $\Gamma_k\ \Psi_k\ \Phi_k\ k$
where *cont*: ‹$((\Gamma,\ n \vdash \Psi \triangleright \Phi) \hookrightarrow^{k} (\Gamma_k,\ \delta k + \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k)) \wedge \varrho \in [\![$
$\Gamma_k,\ \delta k + \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}$›
using *f0 assms*(*1*) *Suc.prems* **by** *blast*
then have *fcontinue*: ‹$\exists\,\Gamma_k{}'\ \Psi_k{}'\ \Phi_k{}'\ k'.\ ((\Gamma_k,\ \delta k + \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k) \hookrightarrow^{k'}$
$(\Gamma_k{}',\ \mathit{Suc}\ (\delta k + \mathit{Suc}\ n) \vdash \Psi_k{}' \triangleright \Phi_k{}'))$
$\wedge\ \varrho \in [\![\ \Gamma_k{}',\ \mathit{Suc}\ (\delta k + \mathit{Suc}\ n) \vdash \Psi_k{}' \triangleright \Phi_k{}'$
$]\!]_{config}$›
using *f0 cont instant-index-increase* **by** *blast*
obtain $\Gamma_k{}'\ \Psi_k{}'\ \Phi_k{}'\ k'$ **where** *cont2*: ‹$((\Gamma_k,\ \delta k + \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k) \hookrightarrow^{k'} (\Gamma_k{}',$
$\mathit{Suc}\ (\delta k + \mathit{Suc}\ n) \vdash \Psi_k{}' \triangleright \Phi_k{}'))$
$\wedge\ \varrho \in [\![\ \Gamma_k{}',\ \mathit{Suc}\ (\delta k + \mathit{Suc}\ n) \vdash \Psi_k{}' \triangleright \Phi_k{}'\ ]\!]_{config}$›
using *Suc.prems* **using** *fcontinue cont* **by** *blast*
have *trans*: ‹$(\Gamma,\ n \vdash \Psi \triangleright \Phi) \hookrightarrow^{k\,+\,k'} (\Gamma_k{}',\ \mathit{Suc}\ (\delta k + \mathit{Suc}\ n) \vdash \Psi_k{}' \triangleright \Phi_k{}')$›
using *operational-semantics-trans-generalized cont cont2*
by *blast*
moreover have *suc-assoc*: ‹$\mathit{Suc}\ \delta k + \mathit{Suc}\ n = \mathit{Suc}\ (\delta k + \mathit{Suc}\ n)$›
by *arith*
ultimately show *?case*
proof (*subst suc-assoc*)
show ‹$\exists\,\Gamma_k\ \Psi_k\ \Phi_k\ k.$
$((\Gamma,\ n \vdash \Psi \triangleright \Phi) \hookrightarrow^{k} (\Gamma_k,\ \mathit{Suc}\ (\delta k + \mathit{Suc}\ n) \vdash \Psi_k \triangleright \Phi_k))$

$\wedge\ \varrho \in [\![\ \Gamma_k,\ Suc\ \delta k + Suc\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}$›
using *cont2 local.trans* **by** *auto*
qed
qed
qed

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

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

6.5 Local termination

primrec *measure-interpretation* :: ‹$'\tau$ :: *linordered-field TESL-formula* ⇒ *nat*› (μ)
where
‹$\mu\ [] = (0{::}nat)$›
| ‹$\mu\ (\varphi\ \#\ \Phi) = ($*case* φ *of*
- sporadic - on - ⇒ $1 + \mu\ \Phi$
| *-* ⇒ $2 + \mu\ \Phi)$›

fun *measure-interpretation-config* :: ‹$'\tau$ :: *linordered-field config* ⇒ *nat*› (μ_{config})
where
‹$\mu_{config}\ (\Gamma,\ n \vdash \Psi \triangleright \Phi) = \mu\ \Psi$›

lemma *elimation-rules-strictly-decreasing*:
assumes ‹$(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow_e (\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2)$›
shows ‹$\mu\ \Psi_1 > \mu\ \Psi_2$›
by (*insert assms*, *erule operational-semantics-elim.cases*, *auto*)

lemma *elimation-rules-strictly-decreasing-meas*:
assumes ‹$(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow_e (\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2)$›
shows ‹$(\Psi_2,\ \Psi_1) \in$ *measure* μ›

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

lemma *elimation-rules-strictly-decreasing-meas'*:
  assumes ‹$\mathcal{S}_1 \hookrightarrow_e \mathcal{S}_2$›
  shows ‹$(\mathcal{S}_2, \mathcal{S}_1) \in$ *measure* μ_{config}›
proof −
  from *assms* **obtain** Γ_1 n_1 Ψ_1 Φ_1 **where** *p1*:‹$\mathcal{S}_1 = (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1)$›
    using *measure-interpretation-config.cases* **by** *blast*
  from *assms* **obtain** Γ_2 n_2 Ψ_2 Φ_2 **where** *p2*:‹$\mathcal{S}_2 = (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$›
    using *measure-interpretation-config.cases* **by** *blast*
  from *elimation-rules-strictly-decreasing-meas assms p1 p2*
    have ‹$(\Psi_2, \Psi_1) \in$ *measure* μ› **by** *blast*
  hence ‹$\mu\ \Psi_2 < \mu\ \Psi_1$› **by** *simp*
  hence ‹$\mu_{config}\ (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2) < \mu_{config}\ (\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1)$› **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 ‹*wfP* $(\lambda(\mathcal{S}_1{::}\ 'a :: linordered\text{-}field\ config)\ \mathcal{S}_2.\ (\mathcal{S}_1 \hookrightarrow_e{}^{\leftarrow} \mathcal{S}_2))$›
  proof (*simp add*: *wfP-def*)
    show ‹*wf* $\{((\mathcal{S}_1{::}\ 'a :: linordered\text{-}field\ config), \mathcal{S}_2).\ \mathcal{S}_1 \hookrightarrow_e{}^{\leftarrow} \mathcal{S}_2\}$›
    proof (*rule wf-subset*)
      have ‹*measure* $\mu_{config} = \{\ (\mathcal{S}_2, (\mathcal{S}_1{::}\ 'a :: linordered\text{-}field\ config)).\ \mu_{config}$ $\mathcal{S}_2 < \mu_{config}\ \mathcal{S}_1\ \}$›
        by (*simp add*: *inv-image-def less-eq measure-def*)
        thus ‹$\{((\mathcal{S}_1{::}\ 'a :: linordered\text{-}field\ config), \mathcal{S}_2).\ \mathcal{S}_1 \hookrightarrow_e{}^{\leftarrow} \mathcal{S}_2\} \subseteq$ (*measure* μ_{config})›
      using *elimation-rules-strictly-decreasing-meas' operational-semantics-elim-inv-def* **by** *blast*
    next
      show ‹*wf* (*measure measure-interpretation-config*)› **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
‹*dilating-fun* $(f{::}nat \Rightarrow nat)$ $(r{::}'a{::}linordered\text{-}field\ run)$
$\equiv$ *strict-mono* $f \wedge (f\ 0 = 0) \wedge (\forall\, n.\ f\ n \geq n$
$\wedge\ ((\nexists\, n_0.\ f\ n_0 = n) \longrightarrow (\forall\, c.\ \neg(hamlet\ ((Rep\text{-}run\ r)\ n\ c))))$
$\wedge\ ((\nexists\, n_0.\ f\ n_0 = (Suc\ n)) \longrightarrow (\forall\, c.\ time\ ((Rep\text{-}run\ r)\ (Suc\ n)\ c) = time$
$((Rep\text{-}run\ r)\ n\ c)))$
)›

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 ‹*dilating f sub r* ≡ *dilating-fun f r*
∧ (∀ *n c*. *time* ((*Rep-run sub*) *n c*) = *time* ((*Rep-run r*) (*f n*) *c*))
∧ (∀ *n c*. *hamlet* ((*Rep-run sub*) *n c*) = *hamlet* ((*Rep-run r*) (*f n*) *c*))›

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

definition *is-subrun* ::‹′*a*::*linordered-field run* ⇒ ′*a run* ⇒ *bool*› (**infixl** ≪ *60*)
where
‹*sub* ≪ *r* ≡ (∃ *f*. *dilating f sub r*)›

A *tick-count r c n* is a number of ticks of clock *c* in run *r* upto instant *n*.

definition *tick-count* :: ‹′*a*::*linordered-field run* ⇒ *clock* ⇒ *nat* ⇒ *nat*›
where
‹*tick-count r c n* = *card* {*i*. *i* ≤ *n* ∧ *hamlet* ((*Rep-run r*) *i c*)}›

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

definition *tick-count-strict* :: ‹′*a*::*linordered-field run* ⇒ *clock* ⇒ *nat* ⇒ *nat*›
where
‹*tick-count-strict r c n* = *card* {*i*. *i* < *n* ∧ *hamlet* ((*Rep-run r*) *i c*)}›

definition *contracting-fun*
where ‹*contracting-fun g* ≡ *mono g* ∧ *g 0* = *0* ∧ (∀ *n*. *g n* ≤ *n*)›

definition *contracting*
where
‹*contracting g r sub f* ≡ *contracting-fun g*
∧ (∀ *n c k*. *f* (*g n*) ≤ *k* ∧ *k* ≤ *n*
⟶ *time* ((*Rep-run r*) *k c*) = *time* ((*Rep-run sub*) (*g n*) *c*))
∧ (∀ *n c k*. *f* (*g n*) < *k* ∧ *k* ≤ *n*
⟶ ¬ *hamlet* ((*Rep-run r*) *k c*))›

definition ‹*dil-inverse f*::(*nat* ⇒ *nat*) ≡ (λ*n*. *Max* {*i*. *f i* ≤ *n*})›

end

7.1.2 Stuttering Lemmas

theory *StutteringLemmas*

imports *StutteringDefs*

begin

lemma *bounded-suc-ind*:
assumes ‹⋀*k*. *k* < *m* ⟹ *P* (*Suc* (*z* + *k*)) = *P* (*z* + *k*)›
shows ‹*k* < *m* ⟹ *P* (*Suc* (*z* + *k*)) = *P z*›
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 ‹$\exists n_0.\ f\ n_0 = n$›
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 ‹$\{k.\ f\ m < k \wedge k < f\ n \wedge$ *hamlet* ((*Rep-run r*) *k c*)$\}$
$= image\ f\ \{k.\ m < k \wedge k < n \wedge$ *hamlet* ((*Rep-run r*) (*f k*) *c*)$\}$›
(**is** ‹*?IMG* = *image f ?SET*›)
proof
{ **fix** *k* **assume** *h*:‹$k \in$ *?IMG*›
from *h* **obtain** k_0 **where** *k0prop*:‹$f\ k_0 = k \wedge$ *hamlet* ((*Rep-run r*) ($f\ k_0$) *c*)›
using *ticks-image*[*OF assms*] **by** *blast*
with *h* **have** ‹$k \in$ *image f ?SET*› **using** *assms dilating-fun-def strict-mono-less*
by *blast*
} **thus** ‹*?IMG* $\subseteq$ *image f ?SET*› **..**
next
{ **fix** *k* **assume** *h*:‹$k \in$ *image f ?SET*›
from *h* **obtain** k_0 **where** *k0prop*:‹$k = f\ k_0 \wedge k_0 \in$ *?SET*› **by** *blast*
hence ‹$k \in$ *?IMG*› **using** *assms* **by** (*simp add*: *dilating-fun-def strict-mono-less*)
} **thus** ‹*image f ?SET* $\subseteq$ *?IMG*› **..**
qed

lemma *dilating-fun-image-left*:
assumes ‹*dilating-fun f r*›
shows ‹$\{k.\ f\ m \leq k \wedge k < f\ n \wedge$ *hamlet* ((*Rep-run r*) *k c*)$\}$

      = *image f* $\{k.\ m \leq k \wedge k < n \wedge$ *hamlet* ((*Rep-run r*) (*f k*) *c*)}›
  (**is** ‹*?IMG* = *image f ?SET*›)
proof
  { fix *k* **assume** *h*:‹*k* ∈ *?IMG*›
    from *h* **obtain** k_0 **where** *k0prop*:‹*f* $k_0 = k \wedge$ *hamlet* ((*Rep-run r*) (*f* k_0) *c*)›
      using *ticks-image*[*OF assms*] **by** *blast*
    with *h* **have** ‹*k* ∈ *image f ?SET*›
      using *assms dilating-fun-def strict-mono-less strict-mono-less-eq* **by** *fastforce*
  } thus ‹*?IMG* ⊆ *image f ?SET*› **..**
next
  { fix *k* **assume** *h*:‹*k* ∈ *image f ?SET*›
    from *h* **obtain** k_0 **where** *k0prop*:‹*k* = *f* $k_0 \wedge k_0 \in$ *?SET*› **by** *blast*
    hence ‹*k* ∈ *?IMG*›
      using *assms dilating-fun-def strict-mono-less strict-mono-less-eq* **by** *fastforce*
  } thus ‹*image f ?SET* ⊆ *?IMG*› **..**
qed

lemma *dilating-fun-image-right*:
  assumes ‹*dilating-fun f r*›
  shows ‹$\{k.\ f\ m < k \wedge k \leq f\ n \wedge$ *hamlet* ((*Rep-run r*) *k c*)}
      = *image f* $\{k.\ m < k \wedge k \leq n \wedge$ *hamlet* ((*Rep-run r*) (*f k*) *c*)}›
  (**is** ‹*?IMG* = *image f ?SET*›)
proof
  { fix *k* **assume** *h*:‹*k* ∈ *?IMG*›
    from *h* **obtain** k_0 **where** *k0prop*:‹*f* $k_0 = k \wedge$ *hamlet* ((*Rep-run r*) (*f* k_0) *c*)›
      using *ticks-image*[*OF assms*] **by** *blast*
    with *h* **have** ‹*k* ∈ *image f ?SET*›
      using *assms dilating-fun-def strict-mono-less strict-mono-less-eq* **by** *fastforce*
  } thus ‹*?IMG* ⊆ *image f ?SET*› **..**
next
  { fix *k* **assume** *h*:‹*k* ∈ *image f ?SET*›
    from *h* **obtain** k_0 **where** *k0prop*:‹*k* = *f* $k_0 \wedge k_0 \in$ *?SET*› **by** *blast*
    hence ‹*k* ∈ *?IMG*›
      using *assms dilating-fun-def strict-mono-less strict-mono-less-eq* **by** *fastforce*
  } thus ‹*image f ?SET* ⊆ *?IMG*› **..**
qed

lemma *dilating-fun-image*:
  assumes ‹*dilating-fun f r*›
  shows ‹$\{k.\ f\ m \leq k \wedge k \leq f\ n \wedge$ *hamlet* ((*Rep-run r*) *k c*)}
      = *image f* $\{k.\ m \leq k \wedge k \leq n \wedge$ *hamlet* ((*Rep-run r*) (*f k*) *c*)}›
  (**is** ‹*?IMG* = *image f ?SET*›)
proof
  { fix *k* **assume** *h*:‹*k* ∈ *?IMG*›
    from *h* **obtain** k_0 **where** *k0prop*:‹*f* $k_0 = k \wedge$ *hamlet* ((*Rep-run r*) (*f* k_0) *c*)›
      using *ticks-image*[*OF assms*] **by** *blast*
    with *h* **have** ‹*k* ∈ *image f ?SET*›
      using *assms dilating-fun-def strict-mono-less-eq* **by** *blast*
  } thus ‹*?IMG* ⊆ *image f ?SET*› **..**

next
  { fix *k* **assume** *h*:‹$k \in$ *image f ?SET*›
    from *h* **obtain** k_0 **where** *k0prop*:‹$k = f\ k_0 \wedge k_0 \in$ *?SET*› **by** *blast*
   hence ‹$k \in$ *?IMG*› **using** *assms* **by** (*simp add*: *dilating-fun-def strict-mono-less-eq*)
  } thus ‹*image f ?SET* $\subseteq$ *?IMG*› **..**
qed

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

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

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

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

moreover from *dilating-fun-image-right*[*OF assms*] **have** ‹*?IMG* = *image f ?SET*› **.**
ultimately show *?thesis* **by** *auto*
qed

lemma *ticks-as-often*:
assumes ‹*dilating-fun f r*›
shows ‹*card* {*p*. $n \leq p \wedge p \leq m \wedge$ *hamlet* ((*Rep-run r*) (*f p*) *c*)}
= *card* {*p*. *f n* $\leq p \wedge p \leq$ *f m* $\wedge$ *hamlet* ((*Rep-run r*) *p c*)}›
(**is** ‹*card ?SET* = *card ?IMG*›)
proof −
from *dilating-fun-injects*[*OF assms*] **have** ‹*inj-on f ?SET*› **.**
moreover have ‹*finite ?SET*› **by** *simp*
from *inj-on-iff-eq-card*[*OF this*] *calculation* **have** ‹*card* (*image f ?SET*) = *card ?SET*› **by** *blast*
moreover from *dilating-fun-image*[*OF assms*] **have** ‹*?IMG* = *image f ?SET*› **.**
ultimately show *?thesis* **by** *auto*
qed

lemma *dilating-injects*:
assumes ‹*dilating f sub r*›
shows ‹*inj-on f A*›
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 ‹*dilating f sub r*›
and ‹*hamlet* ((*Rep-run r*) *n c*)›
shows ‹$\exists\, n_0.\ f\, n_0 = n$›
proof −
from *assms*(*1*) **have** ‹*dilating-fun f r*› **by** (*simp add*: *dilating-def*)
from *ticks-image*[*OF this assms*(*2*)] **show** *?thesis* **.**
qed

lemma *ticks-image-sub'*:
assumes ‹*dilating f sub r*›
and ‹$\exists\, c.$ *hamlet* ((*Rep-run r*) *n c*)›
shows ‹$\exists\, n_0.\ f\, n_0 = n$›
proof −
from *assms*(*1*) **have** ‹*dilating-fun f r*› **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 ‹*dilating f sub r*›
and ‹$\exists\, c.$ *hamlet* ((*Rep-run r*) *k c*)›
and ‹*time* ((*Rep-run r*) *k c*) = τ›

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

TESL operators are preserved by dilation.

lemma *ticks-sub*:
assumes ‹*dilating f sub r*›
shows ‹*hamlet* ((*Rep-run sub*) *n a*) = *hamlet* ((*Rep-run r*) (*f n*) *a*)›
using *assms* **by** (*simp add*: *dilating-def*)

lemma *no-tick-sub*:
assumes ‹*dilating f sub r*›
shows ‹($\nexists\, n_0.\ f\, n_0 = n$) $\longrightarrow$ ¬*hamlet* ((*Rep-run r*) *n a*)›
using *assms dilating-def dilating-fun-def* **by** *blast*

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

definition *opt-lift*::‹('*a* ⇒ '*a*) ⇒ ('*a option* ⇒ '*a option*)›
where
‹*opt-lift f* ≡ λx. *case x of None* ⇒ *None* | *Some y* ⇒ *Some* (*f y*)›

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

lemma *tick-set-sub*:
assumes ‹*dilating f sub r*›
shows ‹{*k*. *hamlet* ((*Rep-run r*) *k c*)} = *image f* {*k*. *hamlet* ((*Rep-run sub*) *k c*)}›
(**is** ‹*?R* = *image f ?S*›)
proof
{ **fix** *k* **assume** *h*:‹*k* ∈ *?R*›
with *no-tick-sub*[*OF assms*] **have** ‹$\exists\, k_0.\ f\, k_0 = k$› **by** *blast*
from *this* **obtain** k_0 **where** *k0prop*:‹$f\, k_0 = k$› **by** *blast*
with *ticks-sub*[*OF assms*] *h* **have** ‹*hamlet* ((*Rep-run sub*) k_0 c)› **by** *blast*
with *k0prop* **have** ‹*k* ∈ *image f ?S*› **by** *blast*
}
thus ‹*?R* ⊆ *image f ?S*› **by** *blast*
next
{ **fix** *k* **assume** *h*:‹*k* ∈ *image f ?S*›
from *this* **obtain** k_0 **where** ‹$f\, k_0 = k \wedge$ *hamlet* ((*Rep-run sub*) k_0 c)› **by** *blast*
with *assms* **have** ‹*k* ∈ *?R*› **using** *ticks-sub* **by** *blast*
}
thus ‹*image f ?S* ⊆ *?R*› **by** *blast*
qed

Strictly monotonous functions preserve the least element.

lemma *Least-strict-mono*:
  assumes ‹*strict-mono f*›
  and ‹$\exists x \in S. \forall y \in S. x \leq y$›
  shows ‹$(LEAST\ y.\ y \in f\ `\ S) = f\ (LEAST\ x.\ x \in S)$›
using *Least-mono*[*OF strict-mono-mono*, *OF assms*] **.**

A non empty set of *nat*s has a least element.

lemma *Least-nat-ex*:
  ‹$(n::nat) \in S \Longrightarrow \exists x \in S. (\forall y \in S. x \leq y)$›
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 ‹*dilating f sub r*›
  and ‹$\exists k::nat.$ *hamlet* ((*Rep-run sub*) *k c*)›
  shows ‹$(LEAST\ k.\ k \in \{t.$ *hamlet* ((*Rep-run r*) *t c*)$\}) = f\ (LEAST\ k.\ k \in \{t.$ *hamlet* ((*Rep-run sub*) *t c*)$\})$›
    (**is** ‹$(LEAST\ k.\ k \in ?R) = f\ (LEAST\ k.\ k \in ?S)$›)
proof −
  from *assms*(*2*) **have** ‹$\exists x.\ x \in ?S$› **by** *simp*
  hence *least*:‹$\exists x \in ?S. \forall y \in ?S. x \leq y$›
    using *Least-nat-ex* **..**
  from *assms*(*1*) **have** ‹*strict-mono f*› **by** (*simp add*: *dilating-def dilating-fun-def*)
  from *Least-strict-mono*[*OF this least*] **have**
    ‹$(LEAST\ y.\ y \in f\ `\ ?S) = f\ (LEAST\ x.\ x \in ?S)$› **.**
  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 ‹*dilating f sub r*›
  and ‹$\exists k.$ *hamlet* ((*Rep-run r*) *k c*)›
  shows ‹$\exists k_0.$ *hamlet* ((*Rep-run sub*) k_0 *c*)›
proof −
  from *assms*(*2*) **obtain** *k* **where** ‹*hamlet* ((*Rep-run r*) *k c*)› **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 ‹*dilating f sub r*›
  and ‹*hamlet* ((*Rep-run r*) *k c*)›
  shows ‹$\exists k_0.\ f\ k_0 = k \land$ *hamlet* ((*Rep-run sub*) k_0 *c*)›
proof −
  from *no-tick-sub*[*OF assms*(*1*)] *assms*(*2*) **have** ‹$\exists k_0.\ f\ k_0 = k$› **by** *blast*
  from *this* **obtain** k_0 **where** ‹$f\ k_0 = k$› **by** *blast*

moreover with *ticks-sub*[*OF assms*(*1*)] *assms*(*2*) **have** ‹*hamlet* ((*Rep-run sub*) k_0 *c*)› **by** *blast*
ultimately show *?thesis* **by** *blast*
qed

A dilating function preserves the tick count on an interval for any clock.

lemma *dilated-ticks-strict*:
assumes ‹*dilating f sub r*›
shows ‹$\{i.\ f\,m < i \wedge i < f\,n \wedge hamlet\ ((Rep\text{-}run\ r)\ i\ c)\}$
$= image\ f\ \{i.\ m < i \wedge i < n \wedge hamlet\ ((Rep\text{-}run\ sub)\ i\ c)\}$›
(**is** ‹*?RUN* = *image f ?SUB*›)
proof
{ **fix** *i* **assume** *h*:‹$i \in$ *?SUB*›
hence ‹$m < i \wedge i < n$› **by** *simp*
hence ‹$f\ m < f\ i \wedge f\ i < (f\ n)$› **using** *assms*
by (*simp add*: *dilating-def dilating-fun-def strict-monoD strict-mono-less-eq*)
moreover from *h* **have** ‹*hamlet* ((*Rep-run sub*) *i c*)› **by** *simp*
hence ‹*hamlet* ((*Rep-run r*) (*f i*) *c*)› **using** *ticks-sub*[*OF assms*] **by** *blast*
ultimately have ‹$f\ i \in$ *?RUN*› **by** *simp*
} **thus** ‹*image f ?SUB* $\subseteq$ *?RUN*› **by** *blast*
next
{ **fix** *i* **assume** *h*:‹$i \in$ *?RUN*›
hence ‹*hamlet* ((*Rep-run r*) *i c*)› **by** *simp*
from *ticks-imp-ticks-subk*[*OF assms this*]
obtain i_0 **where** *i0prop*:‹$f\ i_0 = i \wedge hamlet\ ((Rep\text{-}run\ sub)\ i_0\ c)$› **by** *blast*
with *h* **have** ‹$f\ m < f\ i_0 \wedge f\ i_0 < f\ n$› **by** *simp*
moreover have ‹*strict-mono f*› **using** *assms dilating-def dilating-fun-def* **by** *blast*
ultimately have ‹$m < i_0 \wedge i_0 < n$› **using** *strict-mono-less strict-mono-less-eq* **by** *blast*
with *i0prop* **have** ‹$\exists\, i_0.\ f\ i_0 = i \wedge i_0 \in$ *?SUB*› **by** *blast*
} **thus** ‹*?RUN* $\subseteq$ *image f ?SUB*› **by** *blast*
qed

lemma *dilated-ticks-left*:
assumes ‹*dilating f sub r*›
shows ‹$\{i.\ f\,m \leq i \wedge i < f\,n \wedge hamlet\ ((Rep\text{-}run\ r)\ i\ c)\}$
$= image\ f\ \{i.\ m \leq i \wedge i < n \wedge hamlet\ ((Rep\text{-}run\ sub)\ i\ c)\}$›
(**is** ‹*?RUN* = *image f ?SUB*›)
proof
{ **fix** *i* **assume** *h*:‹$i \in$ *?SUB*›
hence ‹$m \leq i \wedge i < n$› **by** *simp*
hence ‹$f\ m \leq f\ i \wedge f\ i < (f\ n)$› **using** *assms*
by (*simp add*: *dilating-def dilating-fun-def strict-monoD strict-mono-less-eq*)
moreover from *h* **have** ‹*hamlet* ((*Rep-run sub*) *i c*)› **by** *simp*
hence ‹*hamlet* ((*Rep-run r*) (*f i*) *c*)› **using** *ticks-sub*[*OF assms*] **by** *blast*
ultimately have ‹$f\ i \in$ *?RUN*› **by** *simp*
} **thus** ‹*image f ?SUB* $\subseteq$ *?RUN*› **by** *blast*
next

{ **fix** *i* **assume** *h*:‹$i \in$ *?RUN*›
hence ‹*hamlet* ((*Rep-run* *r*) *i* *c*)› **by** *simp*
from *ticks-imp-ticks-subk*[*OF assms this*]
obtain i_0 **where** *i0prop*:‹$f\ i_0 = i \wedge$ *hamlet* ((*Rep-run sub*) i_0 *c*)› **by** *blast*
with *h* **have** ‹$f\ m \leq f\ i_0 \wedge f\ i_0 < f\ n$› **by** *simp*
moreover have ‹*strict-mono f*› **using** *assms dilating-def dilating-fun-def* **by** *blast*
ultimately have ‹$m \leq i_0 \wedge i_0 < n$› **using** *strict-mono-less strict-mono-less-eq* **by** *blast*
with *i0prop* **have** ‹$\exists\, i_0.\ f\ i_0 = i \wedge i_0 \in$ *?SUB*› **by** *blast*
} **thus** ‹*?RUN* $\subseteq$ *image f ?SUB*› **by** *blast*
qed

lemma *dilated-ticks-right*:
assumes ‹*dilating f sub r*›
shows ‹$\{i.\ f\ m < i \wedge i \leq f\ n \wedge$ *hamlet* ((*Rep-run r*) *i c*)$\}$
= *image f* $\{i.\ m < i \wedge i \leq n \wedge$ *hamlet* ((*Rep-run sub*) *i c*)$\}$›
(**is** ‹*?RUN* = *image f ?SUB*›)
proof
{ **fix** *i* **assume** *h*:‹$i \in$ *?SUB*›
hence ‹$m < i \wedge i \leq n$› **by** *simp*
hence ‹$f\ m < f\ i \wedge f\ i \leq (f\ n)$› **using** *assms*
by (*simp add*: *dilating-def dilating-fun-def strict-monoD strict-mono-less-eq*)
moreover from *h* **have** ‹*hamlet* ((*Rep-run sub*) *i c*)› **by** *simp*
hence ‹*hamlet* ((*Rep-run r*) (*f i*) *c*)› **using** *ticks-sub*[*OF assms*] **by** *blast*
ultimately have ‹$f\ i \in$ *?RUN*› **by** *simp*
} **thus** ‹*image f ?SUB* $\subseteq$ *?RUN*› **by** *blast*
next
{ **fix** *i* **assume** *h*:‹$i \in$ *?RUN*›
hence ‹*hamlet* ((*Rep-run r*) *i c*)› **by** *simp*
from *ticks-imp-ticks-subk*[*OF assms this*]
obtain i_0 **where** *i0prop*:‹$f\ i_0 = i \wedge$ *hamlet* ((*Rep-run sub*) i_0 *c*)› **by** *blast*
with *h* **have** ‹$f\ m < f\ i_0 \wedge f\ i_0 \leq f\ n$› **by** *simp*
moreover have ‹*strict-mono f*› **using** *assms dilating-def dilating-fun-def* **by** *blast*
ultimately have ‹$m < i_0 \wedge i_0 \leq n$› **using** *strict-mono-less strict-mono-less-eq* **by** *blast*
with *i0prop* **have** ‹$\exists\, i_0.\ f\ i_0 = i \wedge i_0 \in$ *?SUB*› **by** *blast*
} **thus** ‹*?RUN* $\subseteq$ *image f ?SUB*› **by** *blast*
qed

lemma *dilated-ticks*:
assumes ‹*dilating f sub r*›
shows ‹$\{i.\ f\ m \leq i \wedge i \leq f\ n \wedge$ *hamlet* ((*Rep-run r*) *i c*)$\}$
= *image f* $\{i.\ m \leq i \wedge i \leq n \wedge$ *hamlet* ((*Rep-run sub*) *i c*)$\}$›
(**is** ‹*?RUN* = *image f ?SUB*›)
proof
{ **fix** *i* **assume** *h*:‹$i \in$ *?SUB*›
hence ‹$m \leq i \wedge i \leq n$› **by** *simp*

hence ‹$f\ m \leq f\ i \wedge f\ i \leq (f\ n)$›
using *assms* **by** (*simp add*: *dilating-def dilating-fun-def strict-mono-less-eq*)
moreover from *h* **have** ‹*hamlet* ((*Rep-run sub*) *i c*)› **by** *simp*
hence ‹*hamlet* ((*Rep-run r*) (*f i*) *c*)› **using** *ticks-sub*[*OF assms*] **by** *blast*
ultimately have ‹$f\ i \in ?RUN$› **by** *simp*
} **thus** ‹*image f ?SUB* $\subseteq$ *?RUN*› **by** *blast*
next
{ **fix** *i* **assume** *h*:‹$i \in ?RUN$›
hence ‹*hamlet* ((*Rep-run r*) *i c*)› **by** *simp*
from *ticks-imp-ticks-subk*[*OF assms this*]
obtain i_0 **where** *i0prop*:‹$f\ i_0 = i \wedge$ *hamlet* ((*Rep-run sub*) i_0 *c*)› **by** *blast*
with *h* **have** ‹$f\ m \leq f\ i_0 \wedge f\ i_0 \leq f\ n$› **by** *simp*
moreover have ‹*strict-mono f*› **using** *assms dilating-def dilating-fun-def* **by** *blast*
ultimately have ‹$m \leq i_0 \wedge i_0 \leq n$› **using** *strict-mono-less-eq* **by** *blast*
with *i0prop* **have** ‹$\exists\, i_0.\ f\ i_0 = i \wedge i_0 \in ?SUB$› **by** *blast*
} **thus** ‹*?RUN* $\subseteq$ *image f ?SUB*› **by** *blast*
qed

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

lemma *empty-dilated-prefix*:
assumes ‹*dilating f sub r*›
and ‹$n < f\ 0$›
shows ‹$\neg$ *hamlet* ((*Rep-run r*) *n c*)›
proof −
from *assms* **have** *False* **by** (*simp add*: *dilating-def dilating-fun-def*)
thus *?thesis* **..**
qed

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

corollary *dilated-prefix*:
assumes ‹*dilating f sub r*›

```
  shows  ‹{i. i ≤ f n ∧ hamlet ((Rep-run r) i c)}
          = image f {i. i ≤ n ∧ hamlet ((Rep-run sub) i c)}›
proof –
  have ‹{i. 0 ≤ i ∧ i ≤ f n ∧ hamlet ((Rep-run r) i c)}
        = image f {i. 0 ≤ i ∧ i ≤ n ∧ hamlet ((Rep-run sub) i c)}›
    using dilated-ticks[OF assms] empty-dilated-prefix'[OF assms] by blast
  thus ?thesis by simp
qed

corollary dilated-strict-prefix:
  assumes ‹dilating f sub r›
  shows  ‹{i. i < f n ∧ hamlet ((Rep-run r) i c)}
          = image f {i. i < n ∧ hamlet ((Rep-run sub) i c)}›
proof –
  from assms have dil:‹dilating-fun f r› unfolding dilating-def by simp
  from dil have f0:‹f 0 = 0›  using dilating-fun-def by blast
  from dilating-fun-image-left[OF dil, of ‹0› ‹n› ‹c›]
  have ‹{i. f 0 ≤ i ∧ i < f n ∧ hamlet ((Rep-run r) i c)}
        = image f {i. 0 ≤ i ∧ i < n ∧ hamlet ((Rep-run r) (f i) c)}› .
  hence ‹{i. i < f n ∧ hamlet ((Rep-run r) i c)}
        = image f {i. i < n ∧ hamlet ((Rep-run r) (f i) c)}›
    using f0 by simp
  also have ‹... = image f {i. i < n ∧ hamlet ((Rep-run sub) i c)}›
    using assms dilating-def by blast
  finally show ?thesis by simp
qed
```

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

```
lemma nat-sing-prop:
  ‹{i::nat. i = k ∧ P(i)} = {i::nat. i = k ∧ P(k)}›
by auto
```

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

```
lemma tick-count-is-fun[code]:‹tick-count r c n = run-tick-count r c n›
proof (induction n)
  case 0
    have ‹tick-count r c 0 = card {i. i ≤ 0 ∧ hamlet ((Rep-run r) i c)}›
      by (simp add: tick-count-def)
    also have ‹... = card {i::nat. i = 0 ∧ hamlet ((Rep-run r) 0 c)}›
      using le-zero-eq nat-sing-prop[of ‹0› ‹λi. hamlet ((Rep-run r) i c)›] by simp
    also have ‹... = (if hamlet ((Rep-run r) 0 c) then 1 else 0)› by simp
    also have ‹... = run-tick-count r c 0› by simp
    finally show ?case .
next
  case (Suc k)
    show ?case
    proof (cases ‹hamlet ((Rep-run r) (Suc k) c)›)
      case True
        hence ‹{i. i ≤ Suc k ∧ hamlet ((Rep-run r) i c)} = insert (Suc k) {i. i ≤
```

k ∧ *hamlet* ((*Rep-run r*) *i c*)}›
  by *auto*
 hence ‹*tick-count r c* (*Suc k*) = *Suc* (*tick-count r c k*)›
  by (*simp add*: *tick-count-def*)
 with *Suc.IH* **have** ‹*tick-count r c* (*Suc k*) = *Suc* (*run-tick-count r c k*)› **by** *simp*
 thus *?thesis* **by** (*simp add*: *True*)
next
 case *False*
  hence ‹{*i*. *i* ≤ *Suc k* ∧ *hamlet* ((*Rep-run r*) *i c*)} = {*i*. *i* ≤ *k* ∧ *hamlet* ((*Rep-run r*) *i c*)}›
  using *le-Suc-eq* **by** *auto*
 hence ‹*tick-count r c* (*Suc k*) = *tick-count r c k*› **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*:‹*tick-count-strict r c* (*Suc n*) = *tick-count r c n*›
 unfolding *tick-count-def tick-count-strict-def* **using** *less-Suc-eq-le* **by** *auto*

lemma *tick-count-strict-is-fun*[*code*]:‹*tick-count-strict r c n* = *run-tick-count-strictly r c n*›
proof (*cases* ‹*n* = *0*›)
 case *True*
  hence ‹*tick-count-strict r c n* = *0*› **unfolding** *tick-count-strict-def* **by** *simp*
 also have ‹... = *run-tick-count-strictly r c 0*› **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** ∗:‹*n* = *Suc m*› **by** *blast*
 hence ‹*tick-count-strict r c n* = *tick-count r c m*› **using** *tick-count-strict-suc* **by** *simp*
 also have ‹... = *run-tick-count r c m*› **using** *tick-count-is-fun*[*of* ‹*r*› ‹*c*› ‹*m*›] **.**
 also have ‹... = *run-tick-count-strictly r c* (*Suc m*)› **using** *run-tick-count-strictly.simps*(*2*)[*symmetric*]
.
 finally show *?thesis* **using** ∗ **by** *simp*
qed

lemma *cong-suc-collect*:
 assumes ‹$\bigwedge$*r K n*. *P r K n* = *P′ r K n*›
  and ‹$\bigwedge$*r K n*. *Q r K n* = *Q′ r K n*›
  and ‹$\bigwedge$*r K n*. *Q r K* (*Suc n*) = *P r K n*›
 shows ‹$\bigwedge K_1\ K_2\ n.\ \{r.\ P'\ r\ K_2\ n \leq Q'\ r\ K_1\ n\} = \{r.\ Q'\ r\ K_2\ (Suc\ n) \leq Q'\ r\ K_1\ n\}$›
 using *assms* **by** *auto*

lemma *strictly-precedes-alt-def1*:
‹{ ϱ. ∀ *n*::*nat*. (*run-tick-count* ϱ K_2 *n*) ≤ (*run-tick-count-strictly* ϱ K_1 *n*) }
= { ϱ. ∀ *n*::*nat*. (*run-tick-count-strictly* ϱ K_2 (*Suc n*)) ≤ (*run-tick-count-strictly* ϱ K_1 *n*) }›
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 ‹*P* (*0*::*nat*)›
and ‹⋀*n*. *n* > *0* ⟹ *P n*›
shows ‹*P n*›
using *assms neq0-conv* **by** *blast*

lemma *strictly-precedes-alt-def2*:
‹{ ϱ. ∀ *n*::*nat*. (*run-tick-count* ϱ K_2 *n*) ≤ (*run-tick-count-strictly* ϱ K_1 *n*) }
= { ϱ. (¬*hamlet* ((*Rep-run* ϱ) *0* K_2)) ∧ (∀ *n*::*nat*. (*run-tick-count* ϱ K_2 (*Suc n*)) ≤ (*run-tick-count* ϱ K_1 *n*)) }›
(**is** ‹*?P* = *?P'*›)
proof
{ **fix** *r*::‹*'a run*›
assume ‹*r* ∈ *?P*›
hence ‹∀ *n*::*nat*. (*run-tick-count r* K_2 *n*) ≤ (*run-tick-count-strictly r* K_1 *n*)› **by** *simp*
hence *1*:‹∀ *n*::*nat*. (*tick-count r* K_2 *n*) ≤ (*tick-count-strict r* K_1 *n*)›
using *tick-count-is-fun*[*symmetric*, *of r*] *tick-count-strict-is-fun*[*symmetric*, *of r*] **by** *simp*
hence ‹∀ *n*::*nat*. (*tick-count-strict r* K_2 (*Suc n*)) ≤ (*tick-count-strict r* K_1 *n*)›
using *tick-count-strict-suc*[*symmetric*, *of* ‹*r*› ‹K_2›] **by** *simp*
hence ‹∀ *n*::*nat*. (*tick-count-strict r* K_2 (*Suc* (*Suc n*))) ≤ (*tick-count-strict r* K_1 (*Suc n*))› **by** *simp*
hence ‹∀ *n*::*nat*. (*tick-count r* K_2 (*Suc n*)) ≤ (*tick-count r* K_1 *n*)›
using *tick-count-strict-suc*[*symmetric*, *of* ‹*r*›] **by** *simp*
hence ∗:‹∀ *n*::*nat*. (*run-tick-count r* K_2 (*Suc n*)) ≤ (*run-tick-count r* K_1 *n*)›
by (*simp add*: *tick-count-is-fun*)
from *1* **have** ‹*tick-count r* K_2 *0* <= *tick-count-strict r* K_1 *0*› **by** *simp*
moreover have ‹*tick-count-strict r* K_1 *0* = *0*› **unfolding** *tick-count-strict-def* **by** *simp*
ultimately have ‹*tick-count r* K_2 *0* = *0*› **by** *simp*
hence ‹¬*hamlet* ((*Rep-run r*) *0* K_2)› **unfolding** *tick-count-def* **by** *auto*
with ∗ **have** ‹*r* ∈ *?P'*› **by** *simp*
} **thus** ‹*?P* ⊆ *?P'*› **..**
{ **fix** *r*::‹*'a run*›
assume *h*:‹*r* ∈ *?P'*›
hence ‹∀ *n*::*nat*. (*run-tick-count r* K_2 (*Suc n*)) ≤ (*run-tick-count r* K_1 *n*)› **by** *simp*
hence ‹∀ *n*::*nat*. (*tick-count r* K_2 (*Suc n*)) ≤ (*tick-count r* K_1 *n*)›
by (*simp add*: *tick-count-is-fun*)

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

lemma *run-tick-count-suc*:
‹*run-tick-count r c* (*Suc n*) = (*if hamlet* ((*Rep-run r*) (*Suc n*) *c*)
then Suc (*run-tick-count r c n*)
else run-tick-count r c n)›
by *simp*

corollary *tick-count-suc*:
‹*tick-count r c* (*Suc n*) = (*if hamlet* ((*Rep-run r*) (*Suc n*) *c*)
then Suc (*tick-count r c n*)
else tick-count r c n)›
by (*simp add*: *tick-count-is-fun*)

lemma *card-suc*:‹*card* {*i*. *i* $\leq$ (*Suc n*) $\wedge$ *P i*} = *card* {*i*. *i* $\leq$ *n* $\wedge$ *P i*} + *card* {*i*. *i* = (*Suc n*) $\wedge$ *P i*}›
proof −
have ‹{*i*. *i* $\leq$ *n* $\wedge$ *P i*} $\cap$ {*i*. *i* = (*Suc n*) $\wedge$ *P i*} = {}› **by** *auto*
moreover have ‹{*i*. *i* $\leq$ *n* $\wedge$ *P i*} $\cup$ {*i*. *i* = (*Suc n*) $\wedge$ *P i*} = {*i*. *i* $\leq$ (*Suc n*) $\wedge$ *P i*}› **by** *auto*
moreover have ‹*finite* {*i*. *i* $\leq$ *n* $\wedge$ *P i*}› **by** *simp*
moreover have ‹*finite* {*i*. *i* = (*Suc n*) $\wedge$ *P i*}› **by** *simp*
ultimately show *?thesis* **using** *card-Un-disjoint*[*of* ‹{*i*. *i* $\leq$ *n* $\wedge$ *P i*}› ‹{*i*. *i* = *Suc n* $\wedge$ *P i*}›] **by** *simp*
qed

lemma *card-le-leq*:
assumes ‹*m* < *n*›
shows ‹*card* {*i::nat*. *m* < *i* $\wedge$ *i* $\leq$ *n* $\wedge$ *P i*} = *card* {*i*. *m* < *i* $\wedge$ *i* < *n* $\wedge$ *P i*} + *card* {*i*. *i* = *n* $\wedge$ *P i*}›
proof −
have ‹{*i::nat*. *m* < *i* $\wedge$ *i* < *n* $\wedge$ *P i*} $\cap$ {*i*. *i* = *n* $\wedge$ *P i*} = {}› **by** *auto*
moreover with *assms* **have** ‹{*i::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*}› **by** *auto*

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

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

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

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

lemma *nat-interval-union*:
assumes ‹$m \leq n$›
shows ‹$\{i::nat.\ i \leq n \wedge P\ i\} = \{i::nat.\ i \leq m \wedge P\ i\} \cup \{i::nat.\ m < i \wedge i \leq n \wedge P\ i\}$›
using *assms le-cases nat-less-le* **by** *auto*

lemma *no-tick-before-suc*:
assumes ‹*dilating f sub r*›
and ‹$(f\ n) < k \wedge k < (f\ (Suc\ n))$›
shows ‹$\neg$*hamlet* ((*Rep-run r*) *k c*)›
proof –
from *assms*(*1*) **have** *smf*:‹*strict-mono f*› **by** (*simp add*: *dilating-def dilating-fun-def*)
{ fix *k* **assume** *h*:‹$f\ n < k \wedge k < f\ (Suc\ n) \wedge$ *hamlet* ((*Rep-run r*) *k c*)›
hence ‹$\exists\, k_0.\ f\ k_0 = k$› **using** *assms*(*1*) *dilating-def dilating-fun-def* **by** *blast*
from *this* **obtain** k_0 **where** ‹$f\ k_0 = k$› **by** *blast*
with *h* **have** ‹$f\ n < f\ k_0 \wedge f\ k_0 < f\ (Suc\ n)$› **by** *simp*
hence *False* **using** *smf not-less-eq strict-mono-less* **by** *blast*
} thus *?thesis* **using** *assms*(*2*) **by** *blast*
qed

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

lemma *card-sing-prop*:‹*card* {*i*. *i* = *n* ∧ *P i*} = (*if P n then 1 else 0*)›
proof (*cases* ‹*P n*›)
  case *True*
    hence ‹{*i*. *i* = *n* ∧ *P i*} = {*n*}› **by** (*simp add*: *Collect-conv-if*)
    with ‹*P n*› **show** *?thesis* **by** *simp*
next
  case *False*
    hence ‹{*i*. *i* = *n* ∧ *P i*} = {}› **by** (*simp add*: *Collect-conv-if*)
    with ‹¬*P n*› **show** *?thesis* **by** *simp*
qed

corollary *tick-count-f-suc*:
  assumes ‹*dilating f sub r*›
    shows ‹*tick-count r c* (*f* (*Suc n*)) = *tick-count r c* (*f n*) + (*if hamlet* ((*Rep-run r*) (*f* (*Suc n*)) *c*) *then 1 else 0*)›
using *tick-count-fsuc*[*OF assms*] *card-sing-prop*[*of* ‹*f* (*Suc n*)› ‹λ*k*. *hamlet* ((*Rep-run r*) *k c*)›] **by** *simp*

corollary *tick-count-f-suc-suc*:
  assumes ‹*dilating f sub r*›
    shows ‹*tick-count r c* (*f* (*Suc n*)) = (*if hamlet* ((*Rep-run r*) (*f* (*Suc n*)) *c*)
                                  then Suc (*tick-count r c* (*f n*))
                                  else tick-count r c (*f n*))›
using *tick-count-f-suc*[*OF assms*] **by** *simp*

lemma *tick-count-f-suc-sub*:
  assumes ‹*dilating f sub r*›
    shows ‹*tick-count r c* (*f* (*Suc n*)) = (*if hamlet* ((*Rep-run sub*) (*Suc n*) *c*)
                                  then Suc (*tick-count r c* (*f n*))
                                  else tick-count r c (*f n*))›
using *tick-count-f-suc-suc*[*OF assms*] *assms* **by** (*simp add*: *dilating-def*)

lemma *tick-count-sub*:
  assumes ‹*dilating f sub r*›
    shows ‹*tick-count sub c n* = *tick-count r c* (*f n*)›
proof −
  have ‹*tick-count sub c n* = *card* {*i*. *i* ≤ *n* ∧ *hamlet* ((*Rep-run sub*) *i c*)}›
    using *tick-count-def*[*of* ‹*sub*› ‹*c*› ‹*n*›] **.**
  also have ‹... = *card* (*image f* {*i*. *i* ≤ *n* ∧ *hamlet* ((*Rep-run sub*) *i c*)})›
    using *assms dilating-def dilating-injects*[*OF assms*] **by** (*simp add*: *card-image*)
  also have ‹... = *card* {*i*. *i* ≤ *f n* ∧ *hamlet* ((*Rep-run r*) *i c*)}›
    using *dilated-prefix*[*OF assms*, *symmetric*, *of* ‹*n*› ‹*c*›] **by** *simp*
  also have ‹... = *tick-count r c* (*f n*)›
    using *tick-count-def*[*of* ‹*r*› ‹*c*› ‹*f n*›] **by** *simp*
  finally show *?thesis* **.**
qed

corollary *run-tick-count-sub*:
  assumes ‹*dilating f sub r*›

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

lemma *tick-count-strict-0*:
assumes ‹*dilating f sub r*›
shows ‹*tick-count-strict r c* (*f 0*) = *0*›
proof −
from *assms* **have** ‹*f 0* = *0*› **by** (*simp add*: *dilating-def dilating-fun-def*)
thus *?thesis* **unfolding** *tick-count-strict-def* **by** *simp*
qed

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

lemma *tick-count-strict-stable*:
assumes ‹*dilating f sub r*›
assumes ‹(*f n*) < *k* ∧ *k* < (*f* (*Suc n*))›
shows ‹*tick-count-strict r c k* = *tick-count-strict r c* (*f* (*Suc n*))›
proof −
from *assms*(*1*) **have** *smf*:‹*strict-mono f*› **by** (*simp add*: *dilating-def dilating-fun-def*)
from *assms*(*2*) **have** ‹*f n* < *k*› **by** *simp*
hence ‹∀ *i*. *k* ≤ *i* ⟶ *f n* < *i*› **by** *simp*
with *no-tick-before-suc*[*OF assms*(*1*)] **have**
∗:‹∀ *i*. *k* ≤ *i* ∧ *i* < *f* (*Suc n*) ⟶ ¬*hamlet* ((*Rep-run r*) *i c*)› **by** *blast*
from *tick-count-strict-def* **have** ‹*tick-count-strict r c* (*f* (*Suc n*)) = *card* {*i*. *i* <
f (*Suc n*) ∧ *hamlet* ((*Rep-run r*) *i c*)}› **.**
also have ‹... = *card* {*i*. *i* < *k* ∧ *hamlet* ((*Rep-run r*) *i c*)} + *card* {*i*. *k* ≤ *i* ∧
i < *f* (*Suc n*) ∧ *hamlet* ((*Rep-run r*) *i c*)}›
using *card-mnm′ assms*(*2*) **by** *simp*

  also have ‹... = *card* {*i*. *i* < *k* ∧ *hamlet* ((*Rep-run r*) *i c*)}› **using** ∗ **by** *simp*
  finally show *?thesis* **by** (*simp add*: *tick-count-strict-def*)
qed

lemma *tick-count-strict-sub*:
  assumes ‹*dilating f sub r*›
  shows ‹*tick-count-strict sub c n* = *tick-count-strict r c* (*f n*)›
proof −
  have ‹*tick-count-strict sub c n* = *card* {*i*. *i* < *n* ∧ *hamlet* ((*Rep-run sub*) *i c*)}›
    using *tick-count-strict-def*[*of* ‹*sub*› ‹*c*› ‹*n*›] **.**
  also have ‹... = *card* (*image f* {*i*. *i* < *n* ∧ *hamlet* ((*Rep-run sub*) *i c*)})›
    using *assms dilating-def dilating-injects*[*OF assms*] **by** (*simp add*: *card-image*)
  also have ‹... = *card* {*i*. *i* < *f n* ∧ *hamlet* ((*Rep-run r*) *i c*)}›
    using *dilated-strict-prefix*[*OF assms*, *symmetric*, *of* ‹*n*› ‹*c*›] **by** *simp*
  also have ‹... = *tick-count-strict r c* (*f n*)›
    using *tick-count-strict-def*[*of* ‹*r*› ‹*c*› ‹*f n*›] **by** *simp*
  finally show *?thesis* **.**
qed

lemma *card-prop-mono*:
  assumes ‹*m* ≤ *n*›
    shows ‹*card* {*i*::*nat*. *i* ≤ *m* ∧ *P i*} ≤ *card* {*i*. *i* ≤ *n* ∧ *P i*}›
proof −
  from *assms* **have** ‹{*i*. *i* ≤ *m* ∧ *P i*} ⊆ {*i*. *i* ≤ *n* ∧ *P i*}› **by** *auto*
  moreover have ‹*finite* {*i*. *i* ≤ *n* ∧ *P i*}› **by** *simp*
  ultimately show *?thesis* **by** (*simp add*: *card-mono*)
qed

lemma *mono-tick-count*:
  ‹*mono* (λ *k*. *tick-count r c k*)›
proof
  { **fix** *x y*::*nat*
    assume ‹*x* ≤ *y*›
    from *card-prop-mono*[*OF this*] **have** ‹*tick-count r c x* ≤ *tick-count r c y*›
      unfolding *tick-count-def* **by** *simp*
  } **thus** ‹⋀*x y*. *x* ≤ *y* ⟹ *tick-count r c x* ≤ *tick-count r c y*› **.**
qed

lemma *greatest-prev-image*:
  assumes ‹*dilating f sub r*›
    shows ‹($\nexists\, n_0$. *f* n_0 = *n*) ⟹ (∃ n_p. *f* n_p < *n* ∧ (∀ *k*. *f* n_p < *k* ∧ *k* ≤ *n* ⟶ ($\nexists\, k_0$. *f* k_0 = *k*)))›
proof (*induction n*)
  case *0*
    with *assms* **have** ‹*f 0* = *0*› **by** (*simp add*: *dilating-def dilating-fun-def*)
    thus *?case* **using** *0.prems* **by** *blast*
next
  case (*Suc n*)
  show *?case*

proof (*cases* $\langle \exists\, n_0.\ f\, n_0 = n \rangle$)
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.prems 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 $\langle strict\text{-}mono\ f \rangle$
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 $\langle dilating\ f\ sub\ r \rangle$
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*:$\langle strict\text{-}mono\ f \rangle$ **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 *h*:$\langle Suc\ n > f\ (Suc\ n_p) \rangle$

hence ‹*Suc* n_p < sn_0› **using** ∗∗ *Suc-lessI assms*(*3*) **by** *fastforce*
hence ‹∃ *k*. *k* > n_p ∧ *f k* < *Suc n*› **using** *h* **by** *blast*
with ∗ **have** *False* **using** *smf strict-mono-less* **by** *blast*
} **thus** *?thesis* **using** *not-less* **by** *blast*
qed
hence ‹sn_0 ≤ *Suc* n_p› **using** *assms*(*3*) *smf* **using** *strict-mono-less-eq* **by** *fastforce*
with ∗∗ **show** *?thesis* **by** *simp*
qed

lemma *dil-tick-count*:
assumes ‹*sub* ≪ *r*›
and ‹∀ *n*. *run-tick-count sub a n* ≤ *run-tick-count sub b n*›
shows ‹*run-tick-count r a n* ≤ *run-tick-count r b n*›
proof −
from *assms*(*1*) *is-subrun-def* **obtain** *f* **where** ∗:‹*dilating f sub r*› **by** *blast*
show *?thesis*
proof (*induction n*)
case *0*
from *assms*(*2*) **have** ‹*run-tick-count sub a 0* ≤ *run-tick-count sub b 0*› **..**
with *run-tick-count-sub*[*OF* ∗, *of* - *0*] **have** ‹*run-tick-count r a* (*f 0*) ≤ *run-tick-count r b* (*f 0*)› **by** *simp*
moreover from ∗ **have** ‹*f 0* = *0*› **by** (*simp add*:*dilating-def dilating-fun-def*)
ultimately show *?case* **by** *simp*
next
case (*Suc n′*) **thus** *?case*
proof (*cases* ‹∃ n_0. *f* n_0 = *Suc n′*›)
case *True*
from *this* **obtain** n_0 **where** *fn0*:‹*f* n_0 = *Suc n′*› **by** *blast*
show *?thesis*
proof (*cases* ‹*hamlet* ((*Rep-run sub*) n_0 *a*)›)
case *True*
have *run-tick-count r a* (*f* n_0) ≤ *run-tick-count r b* (*f* n_0)
using *assms*(*2*) *run-tick-count-sub*[*OF* ∗] **by** *simp*
thus *?thesis* **by** (*simp add*: *fn0*)
next
case *False*
hence ‹¬ *hamlet* ((*Rep-run r*) (*Suc n′*) *a*)› **using** ∗ *fn0 ticks-sub* **by** *fastforce*
thus *?thesis* **by** (*simp add*: *Suc.IH le-SucI*)
qed
next
case *False*
thus *?thesis* **using** ∗ *Suc.IH no-tick-sub* **by** *fastforce*
qed
qed
qed

lemma *stutter-no-time*:
assumes ‹*dilating f sub r*›

and ‹$\bigwedge k.\ f\ n < k \wedge k \leq m \Longrightarrow (\nexists k_0.\ f\ k_0 = k)$›
and ‹$m > f\ n$›
shows ‹*time* ((*Rep-run* *r*) *m* *c*) = *time* ((*Rep-run* *r*) (*f* *n*) *c*)›
proof −
from *assms* **have** ‹$\forall k.\ k < m - (f\ n) \longrightarrow (\nexists k_0.\ f\ k_0 = Suc\ ((f\ n) + k))$› **by** *simp*
hence ‹$\forall k.\ k < m - (f\ n)$
$\longrightarrow$ *time* ((*Rep-run* *r*) (*Suc* ((*f* *n*) + *k*)) *c*) = *time* ((*Rep-run* *r*) ((*f* *n*) + *k*) *c*)›
using *assms*(*1*) **by** (*simp add*: *dilating-def dilating-fun-def*)
hence ∗:‹$\forall k.\ k < m - (f\ n) \longrightarrow$ *time* ((*Rep-run* *r*) (*Suc* ((*f* *n*) + *k*)) *c*) = *time* ((*Rep-run* *r*) (*f* *n*) *c*)›
using *bounded-suc-ind*[*of* ‹$m - (f\ n)$› ‹$\lambda k.$ *time* (*Rep-run* *r* *k* *c*)› ‹*f* *n*›] **by** *blast*
from *assms*(*3*) **obtain** m_0 **where** *m0*:‹*Suc* $m_0 = m - (f\ n)$› **using** *Suc-diff-Suc* **by** *blast*
with ∗ **have** ‹*time* ((*Rep-run* *r*) (*Suc* ((*f* *n*) + m_0)) *c*) = *time* ((*Rep-run* *r*) (*f* *n*) *c*)› **by** *auto*
moreover from *m0* **have** ‹*Suc* ((*f* *n*) + m_0) = *m*› **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *time-stuttering*:
assumes ‹*dilating* *f* *sub* *r*›
and ‹*time* ((*Rep-run* *sub*) *n* *c*) = τ›
and ‹$\bigwedge k.\ f\ n < k \wedge k \leq m \Longrightarrow (\nexists k_0.\ f\ k_0 = k)$›
and ‹$m > f\ n$›
shows ‹*time* ((*Rep-run* *r*) *m* *c*) = τ›
proof −
from *assms*(*3*) **have** ‹*time* ((*Rep-run* *r*) *m* *c*) = *time* ((*Rep-run* *r*) (*f* *n*) *c*)›
using *stutter-no-time*[*OF assms*(*1*,*3*,*4*)] **by** *blast*
also from *assms*(*1*,*2*) **have** ‹*time* ((*Rep-run* *r*) (*f* *n*) *c*) = τ› **by** (*simp add*: *dilating-def*)
finally show *?thesis* **.**
qed

lemma *first-time-image*:
assumes ‹*dilating* *f* *sub* *r*›
shows ‹*first-time* *sub* *c* *n* *t* = *first-time* *r* *c* (*f* *n*) *t*›
proof
assume ‹*first-time* *sub* *c* *n* *t*›
with *before-first-time*[*OF this*]
have ∗:‹*time* ((*Rep-run* *sub*) *n* *c*) = *t* ∧ (∀ *m* < *n*. *time*((*Rep-run* *sub*) *m* *c*) < *t*)›
by (*simp add*: *first-time-def*)
moreover have ‹∀ *n* *c*. *time* (*Rep-run* *sub* *n* *c*) = *time* (*Rep-run* *r* (*f* *n*) *c*)›
using *assms*(*1*) **by** (*simp add*: *dilating-def*)
ultimately have ∗∗:‹*time* ((*Rep-run* *r*) (*f* *n*) *c*) = *t* ∧ (∀ *m* < *n*. *time*((*Rep-run* *r*) (*f* *m*) *c*) < *t*)›
by *simp*

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

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

```
  thus ?thesis by simp
qed

lemma not-image-stut:
  assumes ‹dilating f sub r›
      and ‹n0 = Max {i. f i ≤ n}›
      and ‹f n0 < k ∧ k ≤ n›
    shows ‹∄ k0. f k0 = k›
proof –
  from assms(1) have smf:‹strict-mono f›
               and fxge:‹∀ x. f x ≥ x›
    by (auto simp add: dilating-def dilating-fun-def)
  have finite-prefix:‹finite {i. f i ≤ n}› by (simp add: finite-less-ub fxge)
  from assms(1) have ‹f 0 ≤ n› by (simp add: dilating-def dilating-fun-def)
  hence ‹{i. f i ≤ n} ≠ {}› by blast
  from assms(3) fxge have ‹f n0 < n› by linarith
  from assms(2) have ‹∀ x > n0. f x > n› 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 ‹dilating f sub r›
    shows ‹contracting (dil-inverse f) r sub f›
proof –
  from assms have smf:‹strict-mono f›
    and no-img-tick:‹∀ k. (∄ k0. f k0 = k) ⟶ (∀ c. ¬(hamlet ((Rep-run r) k c)))›
    and no-img-time:‹⋀n. (∄ n0. f n0 = (Suc n))
                        ⟶ (∀ c. time ((Rep-run r) (Suc n) c) = time ((Rep-run r)
n c))›
    and fxge:‹∀ x. f x ≥ x› and f0n:‹⋀n. f 0 ≤ n› and f0:‹f 0 = 0›
    by (auto simp add: dilating-def dilating-fun-def)
  have finite-prefix:‹⋀n. finite {i. f i ≤ n}› by (auto simp add: finite-less-ub fxge)
  have prefix-not-empty:‹⋀n. {i. f i ≤ n} ≠ {}› using f0n by blast

  have 1:‹mono (dil-inverse f)›
  proof –
  { fix x::‹nat› and y::‹nat› assume hyp:‹x ≤ y›
    hence inc:‹{i. f i ≤ x} ⊆ {i. f i ≤ y}›
      by (simp add: hyp Collect-mono le-trans)
    from Max-mono[OF inc prefix-not-empty finite-prefix]
      have (dil-inverse f) x ≤ (dil-inverse f) y unfolding dil-inverse-def .
  } thus ?thesis unfolding mono-def by simp
  qed

  from first-dilated-instant[OF smf f0] have 2:‹(dil-inverse f) 0 = 0›
    unfolding dil-inverse-def .

  from fxge have ‹∀ n i. f i ≤ n ⟶ i ≤ n› using le-trans by blast
```

hence *3*:‹$\forall$ *n*. (*dil-inverse f*) $n \leq n$› **using** *Max-in*[*OF finite-prefix prefix-not-empty*]

unfolding *dil-inverse-def* **by** *blast*

from *1 2 3* **have** $*$:‹*contracting-fun* (*dil-inverse f*)› **by** (*simp add*: *contracting-fun-def*)

have *4*:‹$\forall$ *n c k*. *f* ((*dil-inverse f*) *n*) $< k \wedge k \leq n$
$\longrightarrow \neg$ *hamlet* ((*Rep-run r*) *k c*)›
using *not-image-stut*[*OF assms*] *no-img-tick* **unfolding** *dil-inverse-def* **by** *blast*

have *5*:‹($\forall$ *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*))›
proof −
{ **fix** *n c k* **assume** *h*:‹*f* ((*dil-inverse f*) *n*) $\leq k \wedge k \leq n$›
let *?τ* = ‹*time* (*Rep-run sub* ((*dil-inverse f*) *n*) *c*)›
have *tau*:‹*time* (*Rep-run sub* ((*dil-inverse f*) *n*) *c*) = *?τ*› **..**
have *gn*:‹(*dil-inverse f*) *n* = *Max* {*i*. *f i* $\leq$ *n*}› **unfolding** *dil-inverse-def* **..**
from *time-stuttering*[*OF assms tau*, *of k*] *not-image-stut*[*OF assms gn*]
have ‹*time* ((*Rep-run r*) *k c*) = *time* ((*Rep-run sub*) ((*dil-inverse f*) *n*) *c*)›
proof (*cases* ‹*f* ((*dil-inverse f*) *n*) = *k*›)
case *True*
moreover have ‹$\forall$ *n c*. *time* (*Rep-run sub n c*) = *time* (*Rep-run r* (*f n*) *c*)›
using *assms* **by** (*simp add*: *dilating-def*)
ultimately show *?thesis* **by** *simp*
next
case *False*
with *h* **have** ‹*f* (*Max* {*i*. *f i* $\leq$ *n*}) $< k \wedge k \leq n$› **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 ‹$sub \ll r$›
and ‹$sub \in [\![c\ sporadic\ \tau\ on\ c']\!]_{TESL}$›
shows ‹$r \in [\![c\ sporadic\ \tau\ on\ c']\!]_{TESL}$›
proof −
from *assms(1) is-subrun-def* **obtain** *f*
where ‹*dilating f sub r*› **by** *blast*
hence ‹$\forall n\ c.\ time\ ((Rep\text{-}run\ sub)\ n\ c) = time\ ((Rep\text{-}run\ r)\ (f\ n)\ c)$
$\wedge\ hamlet\ ((Rep\text{-}run\ sub)\ n\ c) = hamlet\ ((Rep\text{-}run\ r)\ (f\ n)\ c)$› **by** (*simp add*: *dilating-def*)
moreover from *assms(2)* **have**
‹$sub \in \{r.\ \exists\ n.\ hamlet\ ((Rep\text{-}run\ r)\ n\ c) \wedge time\ ((Rep\text{-}run\ r)\ n\ c') = \tau\}$› **by** *simp*
from *this* **obtain** *k* **where** ‹$time\ ((Rep\text{-}run\ sub)\ k\ c') = \tau \wedge hamlet\ ((Rep\text{-}run\ sub)\ k\ c)$› **by** *auto*
ultimately have ‹$time\ ((Rep\text{-}run\ r)\ (f\ k)\ c') = \tau \wedge hamlet\ ((Rep\text{-}run\ r)\ (f\ k)\ c)$› **by** *simp*
thus *?thesis* **by** *auto*
qed

Implications are preserved in a dilated run.

theorem *implies-sub*:
assumes ‹$sub \ll r$›
and ‹$sub \in [\![c_1\ implies\ c_2]\!]_{TESL}$›
shows ‹$r \in [\![c_1\ implies\ c_2]\!]_{TESL}$›
proof −
from *assms(1) is-subrun-def* **obtain** *f* **where** ‹*dilating f sub r*› **by** *blast*
moreover from *assms(2)* **have**
‹$sub \in \{r.\ \forall n.\ hamlet\ ((Rep\text{-}run\ r)\ n\ c_1) \longrightarrow hamlet\ ((Rep\text{-}run\ r)\ n\ c_2)\}$› **by** *simp*
hence ‹$\forall n.\ hamlet\ ((Rep\text{-}run\ sub)\ n\ c_1) \longrightarrow hamlet\ ((Rep\text{-}run\ sub)\ n\ c_2)$› **by** *simp*
ultimately have ‹$\forall n.\ hamlet\ ((Rep\text{-}run\ r)\ n\ c_1) \longrightarrow hamlet\ ((Rep\text{-}run\ r)\ n\ c_2)$›
using *ticks-imp-ticks-subk ticks-sub* **by** *blast*
thus *?thesis* **by** *simp*
qed

theorem *implies-not-sub*:
assumes ‹$sub \ll r$›
and ‹$sub \in [\![c_1\ implies\ not\ c_2]\!]_{TESL}$›
shows ‹$r \in [\![c_1\ implies\ not\ c_2]\!]_{TESL}$›
proof −
from *assms(1) is-subrun-def* **obtain** *f* **where** ‹*dilating f sub r*› **by** *blast*
moreover from *assms(2)* **have**
‹$sub \in \{r.\ \forall n.\ hamlet\ ((Rep\text{-}run\ r)\ n\ c_1) \longrightarrow \neg\ hamlet\ ((Rep\text{-}run\ r)\ n\ c_2)\}$› **by** *simp*
hence ‹$\forall n.\ hamlet\ ((Rep\text{-}run\ sub)\ n\ c_1) \longrightarrow \neg\ hamlet\ ((Rep\text{-}run\ sub)\ n\ c_2)$› **by** *simp*

ultimately have ‹∀ n. *hamlet* ((*Rep-run* r) n c_1) ⟶ ¬ *hamlet* ((*Rep-run* r) n c_2))›
  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 ‹*sub* ≪ r›
   and ‹*sub* ∈ ⟦c_1 *weakly precedes* c_2⟧$_{TESL}$›
  shows ‹r ∈ ⟦c_1 *weakly precedes* c_2⟧$_{TESL}$›
proof −
 from *assms*(*1*) *is-subrun-def* **obtain** f **where** ∗:‹*dilating f sub r*› **by** *blast*
 from *assms*(*2*) **have**
  ‹*sub* ∈ {r. ∀ n. (*run-tick-count* r c_2 n) ≤ (*run-tick-count* r c_1 n)}› **by** *simp*
 hence ‹∀ n. (*run-tick-count sub* c_2 n) ≤ (*run-tick-count sub* c_1 n)› **by** *simp*
 from *dil-tick-count*[*OF assms*(*1*) *this*] **have** ‹∀ n. (*run-tick-count* r c_2 n) ≤ (*run-tick-count* r c_1 n)› **by** *simp*
 thus *?thesis* **by** *simp*
qed

theorem *strictly-precedes-sub*:
 assumes ‹*sub* ≪ r›
   and ‹*sub* ∈ ⟦c_1 *strictly precedes* c_2⟧$_{TESL}$›
  shows ‹r ∈ ⟦c_1 *strictly precedes* c_2⟧$_{TESL}$›
proof −
 from *assms*(*1*) *is-subrun-def* **obtain** f **where** ∗:‹*dilating f sub r*› **by** *blast*
 from *assms*(*2*) **have** ‹*sub* ∈ { ϱ. ∀ n::*nat*. (*run-tick-count* ϱ c_2 n) ≤ (*run-tick-count-strictly* ϱ c_1 n) }› **by** *simp*
 with *strictly-precedes-alt-def2*[*of* ‹c_2› ‹c_1›] **have**
  ‹*sub* ∈ { ϱ. (¬*hamlet* ((*Rep-run* ϱ) *0* c_2)) ∧ (∀ n::*nat*. (*run-tick-count* ϱ c_2 (*Suc* n)) ≤ (*run-tick-count* ϱ c_1 n)) }›
 by *blast*
 hence ‹(¬*hamlet* ((*Rep-run sub*) *0* c_2)) ∧ (∀ n::*nat*. (*run-tick-count sub* c_2 (*Suc* n)) ≤ (*run-tick-count sub* c_1 n))›
  by *simp*
 hence
  1:‹(¬*hamlet* ((*Rep-run sub*) *0* c_2)) ∧ (∀ n::*nat*. (*tick-count sub* c_2 (*Suc* n)) ≤ (*tick-count sub* c_1 n))›
 by (*simp add*: *tick-count-is-fun*)
 have ‹∀ n::*nat*. (*tick-count* r c_2 (*Suc* n)) ≤ (*tick-count* r c_1 n)›
 proof −
  { **fix** n::*nat*
   have ‹*tick-count* r c_2 (*Suc* n) ≤ *tick-count* r c_1 n›
   proof (*cases* ‹∃ n_0. f n_0 = n›)
    case *True* — n is in the image of f
     from *this* **obtain** n_0 **where** *fn*:‹f n_0 = n› **by** *blast*
     show *?thesis*
     proof (*cases* ‹∃ sn_0. f sn_0 = *Suc* n›)

case *True* — Suc n is in the image of f
from *this* **obtain** sn_0 **where** *fsn*:‹$f\ sn_0 = Suc\ n$› **by** *blast*
with *fn* **have** ‹$sn_0 = Suc\ n_0$› **using** *strict-mono-suc* ∗ *dilating-def dilating-fun-def* **by** *blast*
with *1* **have** ‹*tick-count sub* $c_2\ sn_0 \leq$ *tick-count sub* $c_1\ n_0$› **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 ‹¬*hamlet* ((*Rep-run* r) (*Suc* n) c_2)›
using ∗ **by** (*simp add*: *dilating-def dilating-fun-def*)
hence ‹*tick-count* $r\ c_2$ (*Suc* n) = *tick-count* $r\ c_2\ n$› **by** (*simp add*: *tick-count-suc*)
also have ‹... = *tick-count sub* $c_2\ n_0$› **using** *fn tick-count-sub*[*OF* ∗] **by** *simp*
finally have ‹*tick-count* $r\ c_2$ (*Suc* n) = *tick-count sub* $c_2\ n_0$› **.**
moreover have ‹*tick-count sub* $c_2\ n_0 \leq$ *tick-count sub* c_2 (*Suc* n_0)›
by (*simp add*: *tick-count-suc*)
ultimately have ‹*tick-count* $r\ c_2$ (*Suc* n) $\leq$ *tick-count sub* c_2 (*Suc* n_0)› **by** *simp*
moreover have ‹*tick-count sub* c_2 (*Suc* n_0) $\leq$ *tick-count sub* $c_1\ n_0$› **using** *1* **by** *simp*
ultimately have ‹*tick-count* $r\ c_2$ (*Suc* n) $\leq$ *tick-count sub* $c_1\ n_0$› **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** n_p
where *np-prop*:‹$f\ n_p < n \wedge (\forall k.\ f\ n_p < k \wedge k \leq n \longrightarrow (\nexists k_0.\ f\ k_0 = k))$› **by** *blast*
from *tick-count-latest*[*OF* ∗ *this*] **have** ‹*tick-count* $r\ c_1\ n$ = *tick-count* $r\ c_1\ (f\ n_p)$› **.**
hence *a*:‹*tick-count* $r\ c_1\ n$ = *tick-count sub* $c_1\ n_p$› **using** *tick-count-sub*[*OF* ∗] **by** *simp*
have *b*: ‹*tick-count sub* c_2 (*Suc* n_p) $\leq$ *tick-count sub* $c_1\ n_p$› **using** *1* **by** *simp*
show *?thesis*
proof (*cases* ‹$\exists sn_0.\ f\ sn_0 = Suc\ n$›)
case *True* — Suc n is in the image of f
from *this* **obtain** sn_0 **where** *fsn*:‹$f\ sn_0 = Suc\ n$› **by** *blast*
from *next-non-stuttering*[*OF* ∗ *np-prop this*] **have** *sn-prop*:‹$sn_0 = Suc\ n_p$› **.**
with *b* **have** ‹*tick-count sub* $c_2\ sn_0 \leq$ *tick-count sub* $c_1\ n_p$› **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 ‹¬*hamlet* ((*Rep-run* r) (*Suc* n) c_2)›
using ∗ **by** (*simp add*: *dilating-def dilating-fun-def*)
hence ‹*tick-count* $r\ c_2$ (*Suc* n) = *tick-count* $r\ c_2\ n$› **by** (*simp add*:

tick-count-suc)
also have ‹... = *tick-count sub* c_2 n_p› **using** *np-prop tick-count-sub*[*OF* ∗]
by (*simp add*: *tick-count-latest*[*OF* ∗ *np-prop*])
finally have ‹*tick-count r* c_2 (*Suc n*) = *tick-count sub* c_2 n_p› **.**
moreover have ‹*tick-count sub* c_2 n_p ≤ *tick-count sub* c_2 (*Suc* n_p)›
by (*simp add*: *tick-count-suc*)
ultimately have ‹*tick-count r* c_2 (*Suc n*) ≤ *tick-count sub* c_2 (*Suc* n_p)› **by** *simp*
moreover have ‹*tick-count sub* c_2 (*Suc* n_p) ≤ *tick-count sub* c_1 n_p› **using** *1* **by** *simp*
ultimately have ‹*tick-count r* c_2 (*Suc n*) ≤ *tick-count sub* c_1 n_p› **by** *simp*
thus *?thesis* **using** *np-prop mono-tick-count* **using** *a* **by** *linarith*
qed
qed
} **thus** *?thesis* **..**
qed
moreover from *1* **have** ‹¬*hamlet* ((*Rep-run r*) *0* c_2)›
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 ‹*sub* ≪ *r*›
and ‹*sub* ∈ ⟦ *a time*−*delayed by* $\delta\tau$ *on ms implies b* ⟧$_{TESL}$›
shows ‹*r* ∈ ⟦ *a time*−*delayed by* $\delta\tau$ *on ms implies b* ⟧$_{TESL}$›
proof −
from *assms*(*1*) *is-subrun-def* **obtain** *f* **where** ∗:‹*dilating f sub r*› **by** *blast*
from *assms*(*2*) **have** ‹∀ *n*. *hamlet* ((*Rep-run sub*) *n a*)
⟶ (∀ *m* ≥ *n*. *first-time sub ms m* (*time* ((*Rep-run sub*) *n ms*) + $\delta\tau$)
⟶ *hamlet* ((*Rep-run sub*) *m b*))›
using *TESL-interpretation-atomic.simps*(*5*)[*of* ‹*a*› ‹$\delta\tau$› ‹*ms*› ‹*b*›] **by** *simp*
hence ∗∗:‹∀ n_0. *hamlet* ((*Rep-run r*) (*f* n_0) *a*)
⟶ (∀ m_0 ≥ n_0. *first-time r ms* (*f* m_0) (*time* ((*Rep-run r*) (*f* n_0) *ms*) + $\delta\tau$)
⟶ *hamlet* ((*Rep-run r*) (*f* m_0) *b*)) ›
using *first-time-image*[*OF* ∗] *dilating-def* ∗ **by** *fastforce*
hence ‹∀ *n*. *hamlet* ((*Rep-run r*) *n a*)
⟶ (∀ *m* ≥ *n*. *first-time r ms m* (*time* ((*Rep-run r*) *n ms*) + $\delta\tau$)
⟶ *hamlet* ((*Rep-run r*) *m b*))›
proof −
{ fix *n* **assume** *assm*:‹*hamlet* ((*Rep-run r*) *n a*)›
from *ticks-image-sub*[*OF* ∗ *assm*] **obtain** n_0 **where** *nfn0*:‹*n* = *f* n_0› **by** *blast*
with ∗∗ *assm* **have** *ft0*:
‹(∀ m_0 ≥ n_0. *first-time r ms* (*f* m_0) (*time* ((*Rep-run r*) (*f* n_0) *ms*) + $\delta\tau$)

$\longrightarrow$ *hamlet* ((*Rep-run* *r*) (*f* m_0) *b*))› **by** *blast*
have ‹($\forall m \geq n$. *first-time r ms m* (*time* ((*Rep-run* *r*) *n ms*) + $\delta\tau$)
$\longrightarrow$ *hamlet* ((*Rep-run* *r*) *m b*)) ›
proof −
{ **fix** *m* **assume** *hyp*:‹$m \geq n$›
have ‹*first-time r ms m* (*time* (*Rep-run r n ms*) + $\delta\tau$) $\longrightarrow$ *hamlet* (*Rep-run r m b*)›
proof (*cases* ‹$\exists\, m_0$. *f* m_0 = *m*›)
case *True*
from *this* **obtain** m_0 **where** ‹*m* = *f* m_0› **by** *blast*
moreover have ‹*strict-mono f*› **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* ‹*m* = *0*›)
case *True*
hence ‹*m* = *f 0*› **using** $*$ **by** (*simp add*: *dilating-def dilating-fun-def*)
then show *?thesis* **using** *False* **by** *blast*
next
case *False*
hence ‹$\exists$ *pm*. *m* = *Suc pm*› **by** (*simp add*: *not0-implies-Suc*)
from *this* **obtain** *pm* **where** *mpm*:‹*m* = *Suc pm*› **by** *blast*
hence ‹$\nexists$ pm_0. *f* pm_0 = *Suc pm*› **using** ‹$\nexists\, m_0$. *f* m_0 = *m*› **by** *simp*
with $*$ **have** ‹*time* (*Rep-run r* (*Suc pm*) *ms*) = *time* (*Rep-run r pm ms*)›
using *dilating-def dilating-fun-def* **by** *blast*
hence ‹*time* (*Rep-run r pm ms*) = *time* (*Rep-run r m ms*)› **using** *mpm* **by** *simp*
moreover from *mpm* **have** ‹*pm* < *m*› **by** *simp*
ultimately have ‹$\exists\, m' < m$. *time* (*Rep-run r m' ms*) = *time* (*Rep-run r m ms*)› **by** *blast*
hence ‹¬(*first-time r ms m* (*time* (*Rep-run r n ms*) + $\delta\tau$))›
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 ‹*sub* $\ll$ *r*›
and ‹$r \in [\![$ *time−relation* $\lfloor c_1, c_2 \rfloor \in R$ $]\!]_{TESL}$›
shows ‹$sub \in [\![$ *time−relation* $\lfloor c_1, c_2 \rfloor \in R$ $]\!]_{TESL}$›

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

A time relation is preserved through dilation of a run.

lemma *tagrel-sub'*:
assumes ‹$sub \ll r$›
and ‹$sub \in [\![\ time{-}relation\ \lfloor c_1, c_2 \rfloor \in R\ ]\!]_{TESL}$›
shows ‹$R\ (time\ ((Rep\text{-}run\ r)\ n\ c_1),\ time\ ((Rep\text{-}run\ r)\ n\ c_2))$›
proof –
from *assms*(*1*) *is-subrun-def* **obtain** *f* **where** $*$:‹*dilating f sub r*› **by** *blast*
moreover from *assms*(*2*) *TESL-interpretation-atomic.simps*(*2*) **have**
‹$sub \in \{r.\ \forall n.\ R\ (time\ ((Rep\text{-}run\ r)\ n\ c_1),\ time\ ((Rep\text{-}run\ r)\ n\ c_2))\}$› **by** *blast*
hence *1*:‹$\forall n.\ R\ (time\ ((Rep\text{-}run\ sub)\ n\ c_1),\ time\ ((Rep\text{-}run\ sub)\ n\ c_2))$› **by** *simp*
show *?thesis*
proof (*induction n*)
case *0*
from *1* **have** ‹$R\ (time\ ((Rep\text{-}run\ sub)\ 0\ c_1),\ time\ ((Rep\text{-}run\ sub)\ 0\ c_2))$› **by** *simp*
moreover from $*$ **have** ‹$f\ 0 = 0$› **by** (*simp add*: *dilating-def dilating-fun-def*)
moreover from $*$ **have** ‹$\forall c.\ time\ ((Rep\text{-}run\ sub)\ 0\ c) = time\ ((Rep\text{-}run\ r)\ (f\ 0)\ c)$›
by (*simp add*: *dilating-def*)
ultimately show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
proof (*cases* ‹$\nexists n_0.\ f\ n_0 = Suc\ n$›)
case *True*
with $*$ **have** ‹$\forall c.\ time\ (Rep\text{-}run\ r\ (Suc\ n)\ c) = time\ (Rep\text{-}run\ r\ n\ c)$›
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_0prop:‹$f\ n_0 = Suc\ n$› **by** *blast*
from *1* **have** ‹$R\ (time\ ((Rep\text{-}run\ sub)\ n_0\ c_1),\ time\ ((Rep\text{-}run\ sub)\ n_0\ c_2))$› **by** *simp*

moreover from n_0*prop* $*$ **have** ‹*time* ((*Rep-run sub*) n_0 c_1) = *time* ((*Rep-run r*) (*Suc n*) c_1)›
by (*simp add*: *dilating-def*)
moreover from n_0*prop* $*$ **have** ‹*time* ((*Rep-run sub*) n_0 c_2) = *time* ((*Rep-run r*) (*Suc n*) c_2)›
by (*simp add*: *dilating-def*)
ultimately show *?thesis* **by** *simp*
qed
qed
qed

corollary *tagrel-sub*:
assumes ‹*sub* $\ll$ *r*›
and ‹*sub* $\in$ $[\![$ *time−relation* $\lfloor c_1,c_2 \rfloor$ $\in$ *R* $]\!]_{TESL}$›
shows ‹*r* $\in$ $[\![$ *time−relation* $\lfloor c_1,c_2 \rfloor$ $\in$ *R* $]\!]_{TESL}$›
using *tagrel-sub'*[*OF assms*] **unfolding** *TESL-interpretation-atomic.simps*(*3*) **by** *simp*

theorem *kill-sub*:
assumes ‹*sub* $\ll$ *r*›
and ‹*sub* $\in$ $[\![$ c_1 *kills* c_2 $]\!]_{TESL}$›
shows ‹*r* $\in$ $[\![$ c_1 *kills* c_2 $]\!]_{TESL}$›
proof −
from *assms*(*1*) *is-subrun-def* **obtain** *f* **where** $*$:‹*dilating f sub r*› **by** *blast*
from *assms*(*2*) *TESL-interpretation-atomic.simps*(*8*) **have**
‹$\forall$ *n*. *hamlet* (*Rep-run sub n* c_1) $\longrightarrow$ ($\forall$ *m*$\geq$*n*. $\neg$ *hamlet* (*Rep-run sub m* c_2))›
by *simp*
hence *1*:‹$\forall$ *n*. *hamlet* (*Rep-run r* (*f n*) c_1) $\longrightarrow$ ($\forall$ *m*$\geq$*n*. $\neg$ *hamlet* (*Rep-run r* (*f m*) c_2))›
using *ticks-sub*[*OF* $*$] **by** *simp*
hence ‹$\forall$ *n*. *hamlet* (*Rep-run r* (*f n*) c_1) $\longrightarrow$ ($\forall$ *m*$\geq$ (*f n*). $\neg$ *hamlet* (*Rep-run r m* c_2))›
proof −
{ fix *n* **assume** ‹*hamlet* (*Rep-run r* (*f n*) c_1)›
with *1* **have** *2*:‹$\forall$ *m* $\geq$ *n*. $\neg$ *hamlet* (*Rep-run r* (*f m*) c_2)› **by** *simp*
have ‹$\forall$ *m*$\geq$ (*f n*). $\neg$ *hamlet* (*Rep-run r m* c_2)›
proof −
{ fix *m* **assume** *h*:‹*m* $\geq$ *f n*›
have ‹$\neg$ *hamlet* (*Rep-run r m* c_2)›
proof (*cases* ‹$\exists$ m_0. *f* m_0 = *m*›)
case *True*
from *this* **obtain** m_0 **where** *fm0*:‹*f* m_0 = *m*› **by** *blast*
hence ‹m_0 $\geq$ *n*›
using $*$ *dilating-def dilating-fun-def h strict-mono-less-eq* **by** *fastforce*
with *2* **show** *?thesis* **using** *fm0* **by** *blast*
next
case *False*
thus *?thesis* **using** *ticks-image-sub'*[*OF* $*$] **by** *blast*
qed

```
      } thus ?thesis by simp
    qed
  } thus ?thesis by simp
 qed
 hence ‹∀ n. hamlet (Rep-run r n c₁) ⟶ (∀ m ≥ n. ¬ hamlet (Rep-run r m c₂))›
   using ticks-imp-ticks-subk[OF *] by blast
 thus ?thesis using TESL-interpretation-atomic.simps(8) by blast
qed

lemma atomic-sub:
 assumes ‹sub ≪ r›
     and ‹sub ∈ ⟦ φ ⟧TESL›
   shows ‹r ∈ ⟦ φ ⟧TESL›
proof (cases φ)
 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 ‹sub ≪ r›
   shows ‹sub ∈ ⟦⟦ S ⟧⟧TESL ⟹ r ∈ ⟦⟦ S ⟧⟧TESL›
proof (induction S)
 case Nil
   thus ?case by simp
next
 case (Cons a s)
   from Cons.prems have sa:‹sub ∈ ⟦ a ⟧TESL› and sb:‹sub ∈ ⟦⟦ s ⟧⟧TESL›
     using TESL-interpretation-image by simp+
```

from *Cons.IH*[*OF sb*] **have** ‹$r \in [\![[\![s]\!]]\!]_{TESL}$› **.**
moreover from *atomic-sub*[*OF assms(1) sa*] **have** ‹$r \in [\![a]\!]_{TESL}$› **.**
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.