A Formal Development of a Polychronous Polytimed Coordination Language

Hai NGuyen Van   Frederic Boulanger   Burkhart Wolff

April 10, 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 data-flow networks, discrete event models and so on, as illustrated in Figure 1.1. This raises the interest in architectural composition languages that allow for "bolting the respective sub-models together", along their various interfaces, and specifying the various ways of collaboration and coordination [2].

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

- events may occur in different sub-systems at unrelated times, leading to *polychronous* systems, which do not necessarily have a common base clock,
- the behavior of the sub-systems is observed only at a series of discrete instants, and time coordination has to take this *discretization* into account,
- the instants at which a system is observed may be arbitrary and should not change its behavior (*stuttering invariance*),
- coordination between subsystems involves causality, so the occurrence of an event may enforce the occurrence of other events, possibly after a certain duration has elapsed or an event has occurred a given number of times,

- the domain of time (discrete, rational, continuous,. . .) may be different in the subsystems, leading to *polytimed* systems,
- the time frames of different sub-systems may be related (for instance, time in a GPS satellite and in a GPS receiver on Earth are related although they are not the same).

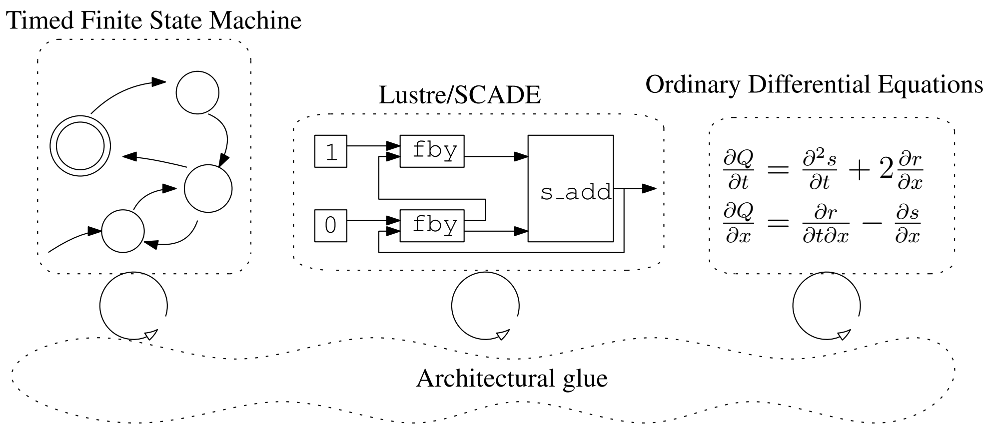

Figure 1.1: A Heterogeneous Timed System Model

In order to tackle the heterogeneous nature of the subsystems, we abstract their behavior as clocks. Each clock models an event – something that can occur or not at a given time. This time is measured in a time frame associated with each clock, and the nature of time (integer, rational, real or any type with a linear order) is specific to each clock. When the event associated with a clock occurs, the clock ticks. In order to support any kind of behavior for the subsystems, we are only interested in specifying what we can observe at a series of discrete instants. There are two constraints on observations: a clock may tick only at an observation instant, and the time on any clock cannot decrease from an instant to the next one. However, it is always possible to add arbitrary observation instants, which allows for stuttering and modular composition of systems. As a consequence, the key concept of our setting is the notion of a clock-indexed Kripke model: $\Sigma^\infty = \mathbb{N} \to \mathcal{K} \to (\mathbb{B} \times \mathcal{T})$, 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 that every time clock `c1` ticks, `c2` must have a tick at the first instant where the time on `m` is `d` later than it was when `c1` had ticked. This means that every tick on `c1` is followed by a tick on `c2` after a delay `d` measured on the time scale of 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.
- Tag constants are just constants of a type which denotes the metric time space.

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

datatype $'\tau$ *tag-const* =
  TConst $'\tau$ (τ_{cst})

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, tags must belong to 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* ::(*field*)*field*
begin
fun *inverse-tag-const*
where ⟨*inverse* (τ_{cst} *t*) = τ_{cst} (*inverse t*)⟩

fun *divide-tag-const*
where ⟨*divide* (τ_{cst} t_1) (τ_{cst} t_2) = τ_{cst} (*divide* t_1 t_2)⟩

fun *uminus-tag-const*
where ⟨*uminus* (τ_{cst} *t*) = τ_{cst} (*uminus t*)⟩

fun *minus-tag-const*
where ⟨*minus* (τ_{cst} t_1) (τ_{cst} t_2) = τ_{cst} (*minus* t_1 t_2)⟩

definition ‹*one-tag-const* ≡ τ_{cst} *1*›

fun *times-tag-const*
  where ‹*times* (τ_{cst} t_1) (τ_{cst} t_2) = τ_{cst} (*times* t_1 t_2)›

definition ‹*zero-tag-const* ≡ τ_{cst} *0*›

fun *plus-tag-const*
  where ‹*plus* (τ_{cst} t_1) (τ_{cst} t_2) = τ_{cst} (*plus* t_1 t_2)›

instance proof

Multiplication is associative.

  fix *a*::‹′τ::*field tag-const*› **and** *b*::‹′τ::*field tag-const*›
                and *c*::‹′τ::*field tag-const*›
  obtain *u v w* **where** ‹*a* = τ_{cst} *u*› **and** ‹*b* = τ_{cst} *v*› **and** ‹*c* = τ_{cst} *w*›
    using *tag-const.exhaust* **by** *metis*
  thus ‹*a* ∗ *b* ∗ *c* = *a* ∗ (*b* ∗ *c*)›
    by (*simp add*: *TESL.times-tag-const.simps*)
next

Multiplication is commutative.

  fix *a*::‹′τ::*field tag-const*› **and** *b*::‹′τ::*field tag-const*›
  obtain *u v* **where** ‹*a* = τ_{cst} *u*› **and** ‹*b* = τ_{cst} *v*› **using** *tag-const.exhaust* **by** *metis*
  thus ‹ *a* ∗ *b* = *b* ∗ *a*›
    by (*simp add*: *TESL.times-tag-const.simps*)
next

One is neutral for multiplication.

  fix *a*::‹′τ::*field tag-const*›
  obtain *u* **where** ‹*a* = τ_{cst} *u*› **using** *tag-const.exhaust* **by** *blast*
  thus ‹*1* ∗ *a* = *a*›
    by (*simp add*: *TESL.times-tag-const.simps one-tag-const-def*)
next

Addition is associative.

  fix *a*::‹′τ::*field tag-const*› **and** *b*::‹′τ::*field tag-const*›
                and *c*::‹′τ::*field tag-const*›
  obtain *u v w* **where** ‹*a* = τ_{cst} *u*› **and** ‹*b* = τ_{cst} *v*› **and** ‹*c* = τ_{cst} *w*›
    using *tag-const.exhaust* **by** *metis*
  thus ‹*a* + *b* + *c* = *a* + (*b* + *c*)›
    by (*simp add*: *TESL.plus-tag-const.simps*)
next

Addition is commutative.

  fix *a*::‹′τ::*field tag-const*› **and** *b*::‹′τ::*field tag-const*›

obtain u v **where** ‹$a = \tau_{cst}$ u› **and** ‹$b = \tau_{cst}$ v› **using** *tag-const.exhaust* **by** *metis*
thus ‹$a + b = b + a$›
by (*simp add*: *TESL.plus-tag-const.simps*)
next

Zero is neutral for addition.

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

The sum of an element and its opposite is zero.

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

Subtraction is adding the opposite.

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

Distributive property of multiplication over addition.

fix a::‹$'\tau$::*field tag-const*› **and** b::‹$'\tau$::*field tag-const*›
and c::‹$'\tau$::*field tag-const*›
obtain u v w **where** ‹$a = \tau_{cst}$ u› **and** ‹$b = \tau_{cst}$ v› **and** ‹$c = \tau_{cst}$ w›
using *tag-const.exhaust* **by** *metis*
thus ‹$(a + b) * c = a * c + b * c$›
by (*simp add*: *TESL.plus-tag-const.simps*
TESL.times-tag-const.simps
ring-class.ring-distribs(*2*))
next

The neutral elements are distinct.

show ‹(0::($'\tau$::*field tag-const*)) $\neq 1$›
by (*simp add*: *one-tag-const-def zero-tag-const-def*)
next

The product of an element and its inverse is 1.

fix *a*::‹$'\tau$::*field tag-const*› **assume** *h*:‹$a \neq 0$›
obtain *u* **where** ‹$a = \tau_{cst}\ u$› **using** *tag-const.exhaust* **by** *blast*
moreover with *h* **have** ‹$u \neq 0$› **by** (*simp add*: *zero-tag-const-def*)
ultimately show ‹*inverse* $a * a = 1$›
by (*simp add*: *TESL.inverse-tag-const.simps*
TESL.times-tag-const.simps
one-tag-const-def)
next

Dividing is multiplying by the inverse.

fix *a*::‹$'\tau$::*field tag-const*› **and** *b*::‹$'\tau$::*field tag-const*›
obtain *u v* **where** ‹$a = \tau_{cst}\ u$› **and** ‹$b = \tau_{cst}\ v$› **using** *tag-const.exhaust* **by** *metis*
thus ‹*a div b* = *a* ∗ *inverse b*›
by (*simp add*: *TESL.divide-tag-const.simps*
TESL.inverse-tag-const.simps
TESL.times-tag-const.simps
divide-inverse)
next

Zero is its own inverse.

show ‹*inverse* (*0*::($'\tau$::*field tag-const*)) = *0*›
by (*simp add*: *TESL.inverse-tag-const.simps zero-tag-const-def*)
qed

end

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

instantiation *tag-const* :: (*order*)*order*
begin
inductive *less-eq-tag-const* :: ‹$'a$ *tag-const* ⇒ $'a$ *tag-const* ⇒ *bool*›
where
Int-less-eq[*simp*]: ‹$n \leq m \Longrightarrow (TConst\ n) \leq (TConst\ m)$›

definition *less-tag*: ‹(x::$'a$ *tag-const*) $< y \longleftrightarrow (x \leq y) \wedge (x \neq y)$›

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** ‹$x = TConst\ x_0$› **by** *blast*
with *Int-less-eq* **show** ‹$x \leq x$› **by** *simp*
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

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

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

end

end

2.2 Defining Runs

theory *Run*
imports *TESL*

begin

Runs are sequences of instants, and each instant maps a clock to a pair that tells 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 ′τ *instant* = ‹*clock* ⇒ (*bool* × ′τ *tag-const*)›

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

typedef (**overloaded**) ′τ::*linordered-field run* =
‹{ ϱ::*nat* ⇒ ′τ *instant*. ∀ *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*:

‹$\forall$ *c. mono* (λ*n. time* (ϱ *n c*)) $\Longrightarrow$ *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* :: ‹('τ::*linordered-field*) *run* $\Rightarrow$ *clock* $\Rightarrow$ *nat* $\Rightarrow$ *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* :: ‹('τ::*linordered-field*) *run* $\Rightarrow$ *clock* $\Rightarrow$ *nat* $\Rightarrow$ *nat*›
  ($\#_{<}$ - - -)
where
  ‹($\#_{<}$ ϱ K *0*)   = *0*›
| ‹($\#_{<}$ ϱ K (*Suc n*)) = $\#_{\leq}$ ϱ K n›

first-time ϱ K n τ tells whether instant n in run ϱ is the first one where the time on clock K reaches τ.

definition *first-time* :: ‹'*a*::*linordered-field run* $\Rightarrow$ *clock* $\Rightarrow$ *nat* $\Rightarrow$ '*a tag-const*
                    $\Rightarrow$ *bool*›
where
  ‹*first-time* ϱ K n τ $\equiv$ (*time* ((*Rep-run* ϱ) n K) = τ)
                  $\wedge$ ($\nexists$ n'. $n' < n$ $\wedge$ *time* ((*Rep-run* ϱ) n' K) = τ)›

The time on a clock is necessarily less than τ before the first instant at which it reaches 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* (λ*n. 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) = τ›
    using *first-time-def* **by** *blast*
  moreover from *assms* **have** ‹*time* ((*Rep-run* ϱ) m K) $\neq$ τ›
    using *first-time-def* **by** *blast*
  ultimately show *?thesis* **by** *simp*

qed

This leads to an alternate definition of *first-time*:

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

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

3.1 Denotational interpretation for atomic TESL formulae

fun *TESL-interpretation-atomic*
  :: ‹('τ::*linordered-field*) *TESL-atomic* ⇒ 'τ *run set*› ($[\![$ - $]\!]_{TESL}$)
where
 — K_1 *sporadic* τ *on* K_2 means that K_1 should tick at an instant where the time on K_2 is τ.
  ‹$[\![\ K_1\ \textit{sporadic}\ \tau\ \textit{on}\ K_2\ ]\!]_{TESL} =$
   $\{\ \varrho.\ \exists\, n{::}nat.\ \textit{hamlet}\ ((\textit{Rep-run}\ \varrho)\ n\ K_1) \wedge \textit{time}\ ((\textit{Rep-run}\ \varrho)\ n\ K_2) = \tau\ \}$›
 — *time−relation* $\lfloor K_1,\ K_2 \rfloor \in R$ means that at each instant, the time on K_1 and the time on K_2 are in relation R.
 | ‹$[\![\ \textit{time}{-}\textit{relation}\ \lfloor K_1,\ K_2 \rfloor \in R\ ]\!]_{TESL} =$
   $\{\ \varrho.\ \forall\, n{::}nat.\ R\ (\textit{time}\ ((\textit{Rep-run}\ \varrho)\ n\ K_1),\ \textit{time}\ ((\textit{Rep-run}\ \varrho)\ n\ K_2))\ \}$›
 — *master implies slave* means that at each instant at which *master* ticks, *slave* also ticks.
 | ‹$[\![\ \textit{master implies slave}\ ]\!]_{TESL} =$
   $\{\ \varrho.\ \forall\, n{::}nat.\ \textit{hamlet}\ ((\textit{Rep-run}\ \varrho)\ n\ \textit{master}) \longrightarrow \textit{hamlet}\ ((\textit{Rep-run}\ \varrho)\ n\ \textit{slave})\ \}$›

— *master implies not slave* means that at each instant at which *master* ticks, *slave* does not tick.
| ‹⟦ *master implies not slave* ⟧$_{TESL}$ =
{ ϱ. ∀ *n*::*nat*. *hamlet* ((*Rep-run* ϱ) *n master*) ⟶ ¬ *hamlet* ((*Rep-run* ϱ) *n slave*) }›
— *master time−delayed by* $\delta\tau$ *on measuring implies slave* means that at each instant at which *master* ticks, *slave* will ticks after a delay $\delta\tau$ measured on the time scale of *measuring*.
| ‹⟦ *master time−delayed by* $\delta\tau$ *on measuring implies slave* ⟧$_{TESL}$ =
— 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.
{ ϱ. ∀ *n*. *hamlet* ((*Rep-run* ϱ) *n master*) ⟶
(*let measured-time* = *time* ((*Rep-run* ϱ) *n measuring*) *in*
∀ *m* ≥ *n*. *first-time* ϱ *measuring m* (*measured-time* + $\delta\tau$)
⟶ *hamlet* ((*Rep-run* ϱ) *m slave*)
)
}›
— K_1 *weakly precedes* K_2 means that each tick on K_2 must be preceded by or coincide with at least one tick on K_1. Therefore, at each instant n, the number of ticks on K_2 must be less or equal to the number of ticks on K_1.
| ‹⟦ K_1 *weakly precedes* K_2 ⟧$_{TESL}$ =
{ ϱ. ∀ *n*::*nat*. (*run-tick-count* ϱ K_2 *n*) ≤ (*run-tick-count* ϱ K_1 *n*) }›
— K_1 *strictly precedes* K_2 means that each tick on K_2 must be preceded by at least one tick on K_1 at a previous instant. Therefore, at each instant n, the number of ticks on K_2 must be less or equal to the number of ticks on K_1 at instant n − (*1*::′*a*).
| ‹⟦ K_1 *strictly precedes* K_2 ⟧$_{TESL}$ =
{ ϱ. ∀ *n*::*nat*. (*run-tick-count* ϱ K_2 *n*) ≤ (*run-tick-count-strictly* ϱ K_1 *n*) }›
— K_1 *kills* K_2 means that when K_1 ticks, K_2 cannot tick and is not allowed to tick at any further instant.
| ‹⟦ K_1 *kills* K_2 ⟧$_{TESL}$ =
{ ϱ. ∀ *n*::*nat*. *hamlet* ((*Rep-run* ϱ) *n* K_1)
⟶ (∀ *m*≥*n*. ¬ *hamlet* ((*Rep-run* ϱ) *m* K_2)) }›

3.2 Denotational interpretation for TESL formulae

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

fun *TESL-interpretation* :: ‹(′τ::*linordered-field*) *TESL-formula* ⇒ ′τ *run set*›
(⟦⟦ - ⟧⟧$_{TESL}$)
where
‹⟦⟦ [] ⟧⟧$_{TESL}$ = { -. *True* }›
| ‹⟦⟦ φ # Φ ⟧⟧$_{TESL}$ = ⟦ φ ⟧$_{TESL}$ ∩ ⟦⟦ Φ ⟧⟧$_{TESL}$›

lemma *TESL-interpretation-homo*:

$\langle [\![\ \varphi\ ]\!]_{TESL} \cap [\![[\ \Phi\ ]\!]]_{TESL} = [\![[\ \varphi\ \#\ \Phi\ ]\!]]_{TESL}\rangle$
by *simp*

3.2.1 Image interpretation lemma

theorem *TESL-interpretation-image*:
$\langle [\![[\ \Phi\ ]\!]]_{TESL} = \bigcap\ ((\lambda\varphi.\ [\![\ \varphi\ ]\!]_{TESL})\ \text{`}\ set\ \Phi)\rangle$
by (*induction* Φ, *simp+*)

3.2.2 Expansion law

Similar to the expansion laws of lattices.

theorem *TESL-interp-homo-append*:
$\langle [\![[\ \Phi_1\ @\ \Phi_2\ ]\!]]_{TESL} = [\![[\ \Phi_1\ ]\!]]_{TESL} \cap [\![[\ \Phi_2\ ]\!]]_{TESL}\rangle$
by (*induction* Φ_1, *simp*, *auto*)

3.3 Equational laws for the denotational interpretation of TESL formulae

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*

The empty formula is the identity element

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

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

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

lemma *TESL-interp-remdups-absorb*:
$\langle [\![[\ \Phi\ ]\!]]_{TESL} = [\![[\ remdups\ \Phi\ ]\!]]_{TESL} \rangle$
proof (*induction* Φ)
case *Cons*
thus *?case* **using** *TESL-interp-formula-stuttering* **by** *auto*
qed *simp*

lemma *TESL-interp-set-lifting*:
assumes $\langle set\ \Phi = set\ \Phi' \rangle$
shows $\langle [\![[\ \Phi\ ]\!]]_{TESL} = [\![[\ \Phi'\ ]\!]]_{TESL} \rangle$
proof −
have $\langle set\ (remdups\ \Phi) = set\ (remdups\ \Phi') \rangle$
by (*simp add*: *assms*)
moreover have *fxpnt*Φ: $\langle \bigcap\ ((\lambda\varphi.\ [\![\ \varphi\ ]\!]_{TESL})\ `\ set\ \Phi) = [\![[\ \Phi\ ]\!]]_{TESL} \rangle$
by (*simp add*: *TESL-interpretation-image*)
moreover have *fxpnt*Φ': $\langle \bigcap\ ((\lambda\varphi.\ [\![\ \varphi\ ]\!]_{TESL})\ `\ set\ \Phi') = [\![[\ \Phi'\ ]\!]]_{TESL} \rangle$
by (*simp add*: *TESL-interpretation-image*)
moreover have $\langle \bigcap\ ((\lambda\varphi.\ [\![\ \varphi\ ]\!]_{TESL})\ `\ set\ \Phi) = \bigcap\ ((\lambda\varphi.\ [\![\ \varphi\ ]\!]_{TESL})\ `\ set\ \Phi') \rangle$
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*
  hence ‹*set* $(\Phi$ @ $\Phi_r) =$ *set* Φ'› **using** *assms* **by** *blast*
  moreover have ‹$($*set* $\Phi) \cup ($*set* $\Phi_r) =$ *set* Φ'›
    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* Φ'›
    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* Φ'›
    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* Φ_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* Φ_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* Φ) $\subseteq$ *set* Φ› **.**
  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* Φ) $\subseteq$ *set* Φ›
by (*rule filter-is-subset*)

end

Chapter 4

Symbolic Primitives for Building Runs

theory *SymbolicPrimitive*
  imports *Run*

begin

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

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

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

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

4.0.1 Symbolic Primitives for Runs

Tag variables are used to get the time on a clock at a given instant index.

datatype *tag-var* =
  TSchematic ⟨*clock* $*$ *instant-index*⟩ (τ_{var})

datatype $'\tau$ *constr* =

— $c \Downarrow n$ @ τ constrains clock c to have time τ at instant n of the run.
Timestamp ⟨*clock*⟩ ⟨*instant-index*⟩ ⟨$'\tau$ *tag-const*⟩ (- ⇓ - @ -)
— m @ $n \oplus \delta t \Rightarrow s$ constrains clock s to tick at the first instant at which the time on m has increased by δt from the value it had at instant n of the run.
| *TimeDelay* ⟨*clock*⟩ ⟨*instant-index*⟩ ⟨$'\tau$ *tag-const*⟩ ⟨*clock*⟩ (- @ - ⊕ - ⇒ -)
— $c \Uparrow n$ constrains clock c to tick at instant n of the run.
| *Ticks* ⟨*clock*⟩ ⟨*instant-index*⟩ (- ⇑ -)
— $c \neg\Uparrow n$ constrains clock c not to tick at instant n of the run.
| *NotTicks* ⟨*clock*⟩ ⟨*instant-index*⟩ (- ¬⇑ -)
— $c \neg\Uparrow < n$ constrains clock c not to tick before instant n of the run.
| *NotTicksUntil* ⟨*clock*⟩ ⟨*instant-index*⟩ (- ¬⇑ < -)
— $c \neg\Uparrow \geq n$ constrains clock c not to tick at and after instant n of the run.
| *NotTicksFrom* ⟨*clock*⟩ ⟨*instant-index*⟩ (- ¬⇑ ≥ -)
— $\lfloor \tau_1, \tau_2 \rfloor \in R$ constrains tag variables τ_1 and τ_2 to be in relation R.
| *TagArith* ⟨*tag-var*⟩ ⟨*tag-var*⟩ ⟨($'\tau$ *tag-const* × $'\tau$ *tag-const*) ⇒ *bool*⟩ (⌊-, -⌋ ∈ -)
— $\lceil k_1, k_2 \rceil \in R$ constrains counter expressions k_1 and k_2 to be in relation R.
| *TickCntArith* ⟨*cnt-expr*⟩ ⟨*cnt-expr*⟩ ⟨(*nat* × *nat*) ⇒ *bool*⟩ (⌈-, -⌉ ∈ -)
— $k_1 \preceq k_2$ constrains counter expression k_1 to be less or equal to counter expression k_2.
| *TickCntLeq* ⟨*cnt-expr*⟩ ⟨*cnt-expr*⟩ (- ⪯ -)

type-synonym $'\tau$ *system* = ⟨$'\tau$ *constr list*⟩

The abstract machine has configurations composed of:

- the past Γ, which captures choices that have already be made as a list of symbolic primitive constraints on the run;
- the current index n, which is the index of the present instant;
- the present Ψ, which captures the formulae that must be satisfied in the current instant;
- the future Φ, which captures the constraints on the future of the run.

type-synonym $'\tau$ *config* =
⟨$'\tau$ *system* ∗ *instant-index* ∗ $'\tau$ *TESL-formula* ∗ $'\tau$ *TESL-formula*⟩

4.1 Semantics of Primitive Constraints

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

fun *counter-expr-eval* :: ⟨($'\tau$::*linordered-field*) *run* ⇒ *cnt-expr* ⇒ *nat*⟩
($[\![$ - ⊢ - $]\!]_{cntexpr}$)
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*
$::\langle('\tau::\mathit{linordered\text{-}field})\ \mathit{constr} \Rightarrow '\tau\ \mathit{run\ set}\rangle$ ($[\![\ \text{-}\ ]\!]_{prim}$)
where
$\langle[\![\ K \Uparrow n\ ]\!]_{prim} \quad = \{\varrho.\ \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ K)\ \}\rangle$
$|\ \langle[\![\ K\ @\ n_0 \oplus \delta t \Rightarrow K'\ ]\!]_{prim} =$
$\quad \{\varrho.\ \forall\, n \geq n_0.\ \mathit{first\text{-}time}\ \varrho\ K\ n\ (\mathit{time}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n_0\ K) + \delta t)$
$\quad\quad \longrightarrow \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ K')\}\rangle$
$|\ \langle[\![\ K \neg\Uparrow n\ ]\!]_{prim} \quad = \{\varrho.\ \neg \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ K)\ \}\rangle$
$|\ \langle[\![\ K \neg\Uparrow < n\ ]\!]_{prim} \quad = \{\varrho.\ \forall\, i < n.\ \neg\ \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ i\ K)\}\rangle$
$|\ \langle[\![\ K \neg\Uparrow \geq n\ ]\!]_{prim} \quad = \{\varrho.\ \forall\, i \geq n.\ \neg\ \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ i\ K)\ \}\rangle$
$|\ \langle[\![\ K \Downarrow n\ @\ \tau\ ]\!]_{prim} = \{\varrho.\ \mathit{time}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ K) = \tau\ \}\rangle$
$|\ \langle[\![\ \lfloor \tau_{var}(K_1,\ n_1),\ \tau_{var}(K_2,\ n_2)\rfloor \in R\ ]\!]_{prim} =$
$\quad \{\ \varrho.\ R\ (\mathit{time}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n_1\ K_1),\ \mathit{time}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n_2\ K_2))\ \}\rangle$
$|\ \langle[\![\ \lceil e_1,\ e_2\rceil \in R\ ]\!]_{prim} = \{\ \varrho.\ R\ ([\![\ \varrho \vdash e_1\ ]\!]_{cntexpr},\ [\![\ \varrho \vdash e_2\ ]\!]_{cntexpr})\ \}\rangle$
$|\ \langle[\![\ \mathit{cnt\text{-}e}_1 \preceq \mathit{cnt\text{-}e}_2\ ]\!]_{prim} = \{\ \varrho.\ [\![\ \varrho \vdash \mathit{cnt\text{-}e}_1\ ]\!]_{cntexpr} \leq [\![\ \varrho \vdash \mathit{cnt\text{-}e}_2\ ]\!]_{cntexpr}\ \}\rangle$

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

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

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

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

4.1.1 Defining a method for witness construction

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

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

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

fun *time-update*
$::\ \langle \mathit{nat} \Rightarrow \mathit{clock} \Rightarrow ('\tau::\mathit{linordered\text{-}field})\ \mathit{tag\text{-}const} \Rightarrow (\mathit{nat} \Rightarrow '\tau\ \mathit{instant})$
$\quad \Rightarrow (\mathit{nat} \Rightarrow '\tau\ \mathit{instant})\rangle$
where
$\langle \mathit{time\text{-}update}\ n\ K\ \tau\ \varrho = (\lambda n'\ K'.\ \mathit{if}\ K = K' \wedge n \leq n'$

then (*hamlet* (ϱ *n* *K*), τ)
else ϱ *n'* *K'*)›

4.2 Rules and properties of consistence

lemma *context-consistency-preservationI*:
‹*consistent-context* ((γ::('τ::*linordered-field*) *constr*)#Γ) $\Longrightarrow$ *consistent-context* Γ›
unfolding *consistent-context-def* **by** *auto*

— This is very restrictive
inductive *context-independency*
::‹('τ::*linordered-field*) *constr* $\Rightarrow$ 'τ *constr list* $\Rightarrow$ *bool*› (- $\bowtie$ -)
where
NotTicks-independency:
‹($K \Uparrow n$) $\notin$ *set* $\Gamma \Longrightarrow$ ($K \neg\Uparrow n$) $\bowtie \Gamma$›
| *Ticks-independency*:
‹($K \neg\Uparrow n$) $\notin$ *set* $\Gamma \Longrightarrow$ ($K \Uparrow n$) $\bowtie \Gamma$›
| *Timestamp-independency*:
‹($\nexists\tau'.\ \tau' = \tau \wedge (K \Downarrow n\ @\ \tau) \in$ *set* Γ) $\Longrightarrow$ ($K \Downarrow n\ @\ \tau$) $\bowtie \Gamma$›

4.3 Major Theorems

4.3.1 Fixpoint lemma

theorem *symrun-interp-fixpoint*:
‹$\bigcap$ (($\lambda\gamma.\ [\![\ \gamma\ ]\!]_{prim}$) ‘ *set* Γ) = $[\![[\ \Gamma\ ]\!]]_{prim}$›
by (*induction* Γ, *simp+*)

4.3.2 Expansion law

Similar to the expansion laws of lattices

theorem *symrun-interp-expansion*:
‹$[\![[\ \Gamma_1\ @\ \Gamma_2\ ]\!]]_{prim} = [\![[\ \Gamma_1\ ]\!]]_{prim} \cap [\![[\ \Gamma_2\ ]\!]]_{prim}$›
by (*induction* Γ_1, *simp*, *auto*)

4.4 Equational laws for the interpretation of symbolic primitives

4.4.1 General laws

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

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

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

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

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

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

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

4.4.2 Decreasing interpretation of symbolic primitives

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

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

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

qed

lemma *symrun-interp-remdups-absorb*:
$\langle[\![[\; \Gamma \;]\!]]_{prim} = [\![[\; remdups\ \Gamma \;]\!]]_{prim}\rangle$
proof (*induction* Γ)
 case *Cons*
  thus *?case* **using** *symrun-interp-formula-stuttering* **by** *auto*
qed *simp*

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

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

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

lemma *symrun-interp-decreases-add-tail*:
 assumes $\langle set\ \Gamma \subseteq set\ \Gamma'\rangle$
  shows $\langle[\![[\; \Gamma\ @\ [\gamma] \;]\!]]_{prim} \supseteq [\![[\; \Gamma'\ @\ [\gamma] \;]\!]]_{prim}\rangle$
proof −
 from *symrun-interp-decreases-setinc*[*OF assms*] **have** $\langle[\![[\; \Gamma' \;]\!]]_{prim} \subseteq [\![[\; \Gamma \;]\!]]_{prim}\rangle$

.
thus *?thesis* **by** (*simp add*: *symrun-interp-expansion dual-order.trans*)
qed

lemma *symrun-interp-absorb1*:
assumes ‹*set* $\Gamma_1 \subseteq$ *set* Γ_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* Γ_1›
shows ‹$[\![\![\ \Gamma_1 \ @\ \Gamma_2\ ]\!]\!]_{prim} = [\![\![\ \Gamma_1\ ]\!]\!]_{prim}$›
using *symrun-interp-absorb1 symrun-interp-commute assms* **by** *blast*

end

Chapter 5

Operational Semantics

theory *Operational*
imports
SymbolicPrimitive

begin

5.1 Operational steps

abbreviation *uncurry-conf*
$::\langle('\tau::\mathit{linordered\text{-}field})\ \mathit{system} \Rightarrow \mathit{instant\text{-}index} \Rightarrow {'\tau}\ \mathit{TESL\text{-}formula} \Rightarrow {'\tau}\ \mathit{TESL\text{-}formula} \Rightarrow {'\tau}\ \mathit{config}\rangle$ $(\text{-},\ \text{-} \vdash \text{-} \triangleright \text{-}\ 80)$
where
$\langle\Gamma,\ n \vdash \Psi \triangleright \Phi \equiv (\Gamma,\ n,\ \Psi,\ \Phi)\rangle$

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

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

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

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

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

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
‹$\mathcal{C}_1 \hookrightarrow_e{}^{\leftarrow} \mathcal{C}_2 \equiv \mathcal{C}_2 \hookrightarrow_e \mathcal{C}_1$›

5.2 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*:
‹($\mathcal{C}_{next}$ (Γ, $n \vdash [] \triangleright \Phi$)) $\supseteq$ { Γ, *Suc* $n \vdash \Phi \triangleright []$ }›
by (*simp add*: *operational-semantics-step.simps operational-semantics-intro.instant-i*)

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

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

$K_2\rfloor \in R$) # Φ) }›
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 6

Equivalence of Operational and Denotational Semantics

theory *Corecursive-Prop*
  imports
    SymbolicPrimitive
    Operational
    Denotational

begin

6.1 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
  $\langle[\![\ K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2\ ]\!]_{TESL}{}^{\geq\ i} =$
    $\{\ \varrho.\ \exists\, n{\geq}i.\ \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ K_1) = \mathit{True} \wedge \mathit{time}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ K_2) = \tau\ \}\rangle$
  $|\ \langle[\![\ \mathit{time}{-}\mathit{relation}\ \lfloor K_1,\ K_2\rfloor \in R\ ]\!]_{TESL}{}^{\geq\ i} =$
    $\{\ \varrho.\ \forall\, n{\geq}i.\ R\ (\mathit{time}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ K_1),\ \mathit{time}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ K_2))\ \}\rangle$
  $|\ \langle[\![\ \mathit{master\ implies\ slave}\ ]\!]_{TESL}{}^{\geq\ i} =$
    $\{\ \varrho.\ \forall\, n{\geq}i.\ \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ \mathit{master}) \longrightarrow \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ \mathit{slave})\ \}\rangle$
  $|\ \langle[\![\ \mathit{master\ implies\ not\ slave}\ ]\!]_{TESL}{}^{\geq\ i} =$
    $\{\ \varrho.\ \forall\, n{\geq}i.\ \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ \mathit{master}) \longrightarrow \neg\ \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ \mathit{slave})\ \}\rangle$
  $|\ \langle[\![\ \mathit{master\ time}{-}\mathit{delayed\ by}\ \delta\tau\ \mathit{on\ measuring\ implies\ slave}\ ]\!]_{TESL}{}^{\geq\ i} =$
    $\{\ \varrho.\ \forall\, n{\geq}i.\ \mathit{hamlet}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ \mathit{master}) \longrightarrow$
      $(\mathit{let\ measured\text{-}time} = \mathit{time}\ ((\mathit{Rep\text{-}run}\ \varrho)\ n\ \mathit{measuring})\ \mathit{in}$
      $\forall\, m \geq n.\ \mathit{first\text{-}time}\ \varrho\ \mathit{measuring}\ m\ (\mathit{measured\text{-}time} + \delta\tau)$

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

theorem *predicate-Union-unfold*:
‹{ ϱ. $\exists\, n$. $P\ \varrho\ n$} = $\bigcup$ {Y. $\exists\, n$. Y = { ϱ. $P\ \varrho\ n$ }}›
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* $\lfloor K_1,\ K_2 \rfloor \in R$ ⟧$_{TESL}$ = $\bigcap$ {Y. $\exists\, n$::*nat*. Y = ⟦ *time*−*relation* $\lfloor K_1,\ K_2 \rfloor \in R$ ⟧$_{TESL}{}^{\geq\ n}$}›
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*:
$\langle(\forall\, n \geq (n_0{::}nat).\ P\ n) = (P\ n_0 \wedge (\forall\, n \geq Suc\ n_0.\ P\ n))\rangle$
proof −
have $\langle(\forall\, n \geq (n_0{::}nat).\ P\ n) = (\forall\, n.\ (n = n_0 \vee n > n_0) \longrightarrow P\ n)\rangle$ **using** *le-less* **by** *blast*
also have $\langle ... = (P\ n_0 \wedge (\forall\, n > n_0.\ P\ n))\rangle$ **by** *blast*
finally show *?thesis* **using** *Suc-le-eq* **by** *simp*
qed

lemma *exists-nat-expansion*:
$\langle(\exists\, n \geq (n_0{::}nat).\ P\ n) = (P\ n_0 \vee (\exists\, n \geq Suc\ n_0.\ P\ n))\rangle$
proof −
have $\langle(\exists\, n \geq (n_0{::}nat).\ P\ n) = (\exists\, n.\ (n = n_0 \vee n > n_0) \wedge P\ n)\rangle$ **using** *le-less* **by** *blast*
also have $\langle ... = (\exists\, n.\ (P\ n_0) \vee (n > n_0 \wedge P\ n))\rangle$ **by** *blast*
finally show *?thesis* **using** *Suc-le-eq* **by** *simp*
qed

6.2 Coinduction Unfolding Properties

lemma *TESL-interp-stepwise-sporadicon-cst-coind-unfold*:
shows $\langle[\![\ K_1\ sporadic\ \tau\ on\ K_2\ ]\!]_{TESL}{}^{\geq\ n} =$
$[\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2 \Downarrow n\ @\ \tau\ ]\!]_{prim}$
$\cup\ [\![\ K_1\ sporadic\ \tau\ on\ K_2\ ]\!]_{TESL}{}^{\geq\ Suc\ n}\rangle$
proof −
have $\langle\{\ \varrho.\ \exists\, m{\geq}n.\ hamlet\ ((Rep\text{-}run\ \varrho)\ m\ K_1) = True \wedge time\ ((Rep\text{-}run\ \varrho)\ m\ K_2) = \tau\ \}$
$= \{\ \varrho.\ hamlet\ ((Rep\text{-}run\ \varrho)\ n\ K_1) = True \wedge time\ ((Rep\text{-}run\ \varrho)\ n\ K_2) = \tau$
$\vee\ (\exists\, m{\geq}Suc\ n.\ hamlet\ ((Rep\text{-}run\ \varrho)\ m\ K_1) = True \wedge time\ ((Rep\text{-}run\ \varrho)\ m\ K_2) = \tau)\ \}\rangle$
using *Suc-leD not-less-eq-eq* **by** *fastforce*
moreover have $\langle\{\ \varrho.\ hamlet\ ((Rep\text{-}run\ \varrho)\ n\ K_1) = True \wedge time\ ((Rep\text{-}run\ \varrho)\ n\ K_2) = \tau$
$\vee\ (\exists\, m{\geq}Suc\ n.\ hamlet\ ((Rep\text{-}run\ \varrho)\ m\ K_1) = True \wedge time\ ((Rep\text{-}run\ \varrho)\ m\ K_2) = \tau)\ \}$
$= [\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2 \Downarrow n\ @\ \tau\ ]\!]_{prim} \cup [\![\ K_1\ sporadic\ \tau\ on\ K_2\ ]\!]_{TESL}{}^{\geq\ Suc\ n}\rangle$
by (*simp add*: *Collect-conj-eq Collect-disj-eq*)
ultimately show *?thesis* **by** *auto*
qed

lemma *TESL-interp-stepwise-sporadicon-coind-unfold*:
shows $\langle[\![\ K_1\ sporadic\ \tau\ on\ K_2\ ]\!]_{TESL}{}^{\geq\ n} =$
$[\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2 \Downarrow n\ @\ \tau\ ]\!]_{prim}$

$\cup$ $[\![$ K_1 *sporadic* τ *on* K_2 $]\!]_{TESL}{}^{\geq\ Suc\ n}\rangle$
using *TESL-interp-stepwise-sporadicon-cst-coind-unfold* **by** *blast*

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

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

lemma *TESL-interp-stepwise-implies-coind-unfold*:
shows $\langle[\![$ *master implies slave* $]\!]_{TESL}{}^{\geq\ n}$ =
($[\![$ *master* $\neg\Uparrow$ n $]\!]_{prim}$ $\cup$ $[\![$ *master* $\Uparrow$ n $]\!]_{prim}$ $\cap$ $[\![$ *slave* $\Uparrow$ n $]\!]_{prim}$)
$\cap$ $[\![$ *master implies slave* $]\!]_{TESL}{}^{\geq\ Suc\ n}\rangle$
proof −
have $\langle$\{ ϱ. $\forall m{\geq}n$. *hamlet* ((*Rep-run* ϱ) m *master*) $\longrightarrow$ *hamlet* ((*Rep-run* ϱ) m *slave*) }
= { ϱ. *hamlet* ((*Rep-run* ϱ) n *master*) $\longrightarrow$ *hamlet* ((*Rep-run* ϱ) n *slave*) }
$\cap$ { ϱ. $\forall m{\geq}Suc\ n$. *hamlet* ((*Rep-run* ϱ) m *master*) $\longrightarrow$ *hamlet* ((*Rep-run* ϱ) m *slave*) }$\rangle$
using *nat-set-suc*[*of* $\langle n\rangle$ $\langle\lambda x\ y.$ *hamlet* ((*Rep-run* x) y *master*) $\longrightarrow$ *hamlet* ((*Rep-run* x) y *slave*)$\rangle$] **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* $\neg\Uparrow$ *n* $]\!]_{prim}$ ∪ $[\![$ *master* $\Uparrow$ *n* $]\!]_{prim}$ ∩ $[\![$ *slave* $\neg\Uparrow$ *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* $\neg\Uparrow$ *n* $]\!]_{prim}$ ∪ ($[\![$ *master* $\Uparrow$ *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*)) $]\!]_{prim}$
∩ $[\![$ K_1 *weakly precedes* K_2 $]\!]_{TESL}^{\geq\ Suc\ n}$›
proof −
have ‹{ ϱ. ∀ *p*≥*n*. (*run-tick-count* ϱ K_2 *p*) ≤ (*run-tick-count* ϱ K_1 *p*) }
= { ϱ. (*run-tick-count* ϱ K_2 *n*) ≤ (*run-tick-count* ϱ K_1 *n*) }
∩ { ϱ. ∀ *p*≥*Suc n*. (*run-tick-count* ϱ K_2 *p*) ≤ (*run-tick-count* ϱ K_1 *p*) }›
using *nat-set-suc*[*of* ‹*n*› ‹λϱ *n*. (*run-tick-count* ϱ K_2 *n*) ≤ (*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* = ‹$\lambda n\ \varrho$. $\forall\,p{\geq}n$. *hamlet* ((*Rep-run* ϱ) p K_1) $\longrightarrow$ ($\forall\,m{\geq}p$. $\neg$ *hamlet* ((*Rep-run* ϱ) m K_2))›
  let *?ticks* = ‹$\lambda n\ \varrho\ c$. *hamlet* ((*Rep-run* ϱ) n c)›
  let *?dead* = ‹$\lambda n\ \varrho\ c$. $\forall\,m \geq n$. $\neg$*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}
      $\subseteq$ {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}› **by** *blast*
  next
   { fix ϱ::‹$'\tau$::*linordered-field run*›
    assume ‹$\varrho \in$ ({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})›
    hence ‹$\neg$ *?ticks* n ϱ K_1 $\wedge$ *?kills* (*Suc* n) ϱ
        $\vee$ *?ticks* n ϱ K_1 $\wedge$ *?dead* n ϱ K_2› **by** *blast*
    moreover have ‹(($\neg$ *?ticks* n ϱ K_1) $\wedge$ (*?kills* (*Suc* n) ϱ)) $\longrightarrow$ *?kills* n ϱ›
     using *dual-order.antisym not-less-eq-eq* **by** *blast*
    ultimately have ‹*?kills* n ϱ $\vee$ *?ticks* n ϱ K_1 $\wedge$ *?dead* n ϱ K_2› **by** *blast*

hence ‹*?kills n* ϱ› **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 ‹... $= [\![\ K_1\ \neg\Uparrow\ n\ ]\!]_{prim} \cap [\![\ K_1\ \mathit{kills}\ K_2\ ]\!]_{TESL}{}^{\geq\ Suc\ n}$
$\cup\ [\![\ K_1 \Uparrow n\ ]\!]_{prim} \cap [\![\ K_2\ \neg\Uparrow \geq n\ ]\!]_{prim} \cap [\![\ K_1\ \mathit{kills}\ K_2\ ]\!]_{TESL}{}^{\geq\ Suc\ n}$›
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*:
‹$[\![[\![\ \Phi\ ]\!]]\!]_{TESL}{}^{\geq\ n} = \bigcap\ ((\lambda\varphi.\ [\![\ \varphi\ ]\!]_{TESL}{}^{\geq\ n})$ ‘ *set* $\Phi)$›
by (*induction* Φ, *simp*, *auto*)

lemma *TESL-interpretation-stepwise-zero*:
‹$[\![\ \varphi\ ]\!]_{TESL} = [\![\ \varphi\ ]\!]_{TESL}{}^{\geq\ 0}$›
by (*induction* φ, *simp+*)

lemma *TESL-interpretation-stepwise-zero'*:
‹$[\![[\![\ \Phi\ ]\!]]\!]_{TESL} = [\![[\![\ \Phi\ ]\!]]\!]_{TESL}{}^{\geq\ 0}$›
by (*induction* Φ, *simp*, *simp add*: *TESL-interpretation-stepwise-zero*)

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

theorem *TESL-interp-stepwise-composition*:
shows ‹$[\![[\![\ \Phi_1\ @\ \Phi_2\ ]\!]]\!]_{TESL}{}^{\geq\ n} = [\![[\![\ \Phi_1\ ]\!]]\!]_{TESL}{}^{\geq\ n} \cap [\![[\![\ \Phi_2\ ]\!]]\!]_{TESL}{}^{\geq\ n}$›
by (*induction* Φ_1, *simp*, *auto*)

6.3 Interpretation of configurations

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

lemma *HeronConf-interp-composition*:
shows ‹$[\![\ \Gamma_1,\ n \vdash \Psi_1 \triangleright \Phi_1\ ]\!]_{config} \cap [\![\ \Gamma_2,\ n \vdash \Psi_2 \triangleright \Phi_2\ ]\!]_{config}$
$= [\![\ (\Gamma_1\ @\ \Gamma_2),\ n \vdash (\Psi_1\ @\ \Psi_2) \triangleright (\Phi_1\ @\ \Phi_2)\ ]\!]_{config}$›

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*: ‹$([\![\, K_1\ \neg\Uparrow\ n\,]\!]_{prim} \cup [\![\, K_1 \Uparrow n\,]\!]_{prim} \cap [\![\, K_2 \Uparrow n\,]\!]_{prim}) \cap [\![\, K_1\ \mathit{implies}\ K_2\,]\!]_{TESL}{}^{\geq\ Suc\ n} \cap ([\![[\, \Psi\,]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\, \Phi\,]\!]]_{TESL}{}^{\geq\ Suc\ n}) = [\![[\, (K_1\ \mathit{implies}\ K_2)\ \#\ \Psi\,]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\, \Phi\,]\!]]_{TESL}{}^{\geq\ Suc\ n}$›
using *TESL-interp-stepwise-implies-coind-unfold TESL-interpretation-stepwise-cons-morph* **by** *blast*
have ‹$[\![\, K_1\ \neg\Uparrow\ n\,]\!]_{prim} \cap [\![[\, \Gamma\,]\!]]_{prim} \cup [\![\, K_1 \Uparrow n\,]\!]_{prim} \cap [\![[\, (K_2 \Uparrow n)\ \#\ \Gamma$

$]\!]_{prim} = ([\![\ K_1 \neg\Uparrow n \ ]\!]_{prim} \cup [\![\ K_1 \Uparrow n \ ]\!]_{prim} \cap [\![\ K_2 \Uparrow n \ ]\!]_{prim}) \cap [\![[\![\ \Gamma \ ]\!]]_{prim}\rangle$
  by *force*
 then have $\langle[\![\ \Gamma, n \vdash ((K_1 \textit{ implies } K_2) \# \Psi) \triangleright \Phi \ ]\!]_{config} = ([\![\ K_1 \neg\Uparrow n \ ]\!]_{prim} \cap [\![[\![\ \Gamma \ ]\!]]_{prim} \cup [\![\ K_1 \Uparrow n \ ]\!]_{prim} \cap [\![[\![\ (K_2 \Uparrow n) \# \Gamma \ ]\!]]_{prim}) \cap ([\![[\![\ \Psi \ ]\!]]_{TESL}^{\geq \ n} \cap [\![[\![\ (K_1 \textit{ implies } K_2) \# \Phi \ ]\!]]_{TESL}^{\geq \ Suc \ n})\rangle$
  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-implies-not-cases*:
 shows $\langle[\![\ \Gamma, n \vdash ((K_1 \textit{ implies not } K_2) \# \Psi) \triangleright \Phi \ ]\!]_{config}$
  $= [\![\ ((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \textit{ implies not } K_2) \# \Phi) \ ]\!]_{config}$
  $\cup [\![\ ((K_1 \Uparrow n) \# (K_2 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \textit{ implies not } K_2) \# \Phi) \ ]\!]_{config}\rangle$
proof −
 have $\langle[\![\ \Gamma, n \vdash (K_1 \textit{ implies not } K_2) \# \Psi \triangleright \Phi \ ]\!]_{config} = [\![[\![\ \Gamma \ ]\!]]_{prim} \cap [\![[\![\ (K_1 \textit{ implies not } K_2) \# \Psi \ ]\!]]_{TESL}^{\geq \ n} \cap [\![[\![\ \Phi \ ]\!]]_{TESL}^{\geq \ Suc \ n}\rangle$
  by *simp*
 moreover have $\langle[\![\ ((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \textit{ implies not } K_2) \# \Phi) \ ]\!]_{config}$
  $= [\![[\![\ (K_1 \neg\Uparrow n) \# \Gamma \ ]\!]]_{prim} \cap [\![[\![\ \Psi \ ]\!]]_{TESL}^{\geq \ n} \cap [\![[\![\ (K_1 \textit{ implies not } K_2) \# \Phi \ ]\!]]_{TESL}^{\geq \ Suc \ n}\rangle$
  by *simp*
 moreover have $\langle[\![\ ((K_1 \Uparrow n) \# (K_2 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \textit{ implies not } K_2) \# \Phi) \ ]\!]_{config}$
  $= [\![[\![\ ((K_1 \Uparrow n) \# (K_2 \neg\Uparrow n) \# \Gamma) \ ]\!]]_{prim} \cap [\![[\![\ \Psi \ ]\!]]_{TESL}^{\geq \ n} \cap [\![[\![\ (K_1 \textit{ implies not } K_2) \# \Phi \ ]\!]]_{TESL}^{\geq \ Suc \ n}\rangle$
  by *simp*
 ultimately show *?thesis*
 proof −
  have *f1*: $\langle([\![\ K_1 \neg\Uparrow n \ ]\!]_{prim} \cup [\![\ K_1 \Uparrow n \ ]\!]_{prim} \cap [\![\ K_2 \neg\Uparrow n \ ]\!]_{prim}) \cap [\![\ K_1 \textit{ implies not } K_2 \ ]\!]_{TESL}^{\geq \ Suc \ n} \cap ([\![[\![\ \Psi \ ]\!]]_{TESL}^{\geq \ n} \cap [\![[\![\ \Phi \ ]\!]]_{TESL}^{\geq \ Suc \ n})$
   $= [\![[\![\ (K_1 \textit{ implies not } K_2) \# \Psi \ ]\!]]_{TESL}^{\geq \ n} \cap [\![[\![\ \Phi \ ]\!]]_{TESL}^{\geq \ Suc \ n}\rangle$
  using *TESL-interp-stepwise-implies-not-coind-unfold TESL-interpretation-stepwise-cons-morph* **by** *blast*
  have $\langle[\![\ K_1 \neg\Uparrow n \ ]\!]_{prim} \cap [\![[\![\ \Gamma \ ]\!]]_{prim} \cup [\![\ K_1 \Uparrow n \ ]\!]_{prim} \cap [\![[\![\ (K_2 \neg\Uparrow n) \# \Gamma \ ]\!]]_{prim} = ([\![\ K_1 \neg\Uparrow n \ ]\!]_{prim} \cup [\![\ K_1 \Uparrow n \ ]\!]_{prim} \cap [\![\ K_2 \neg\Uparrow n \ ]\!]_{prim}) \cap [\![[\![\ \Gamma \ ]\!]]_{prim}\rangle$
   by *force*
  then have $\langle[\![\ \Gamma, n \vdash ((K_1 \textit{ implies not } K_2) \# \Psi) \triangleright \Phi \ ]\!]_{config}$
   $= ([\![\ K_1 \neg\Uparrow n \ ]\!]_{prim} \cap [\![[\![\ \Gamma \ ]\!]]_{prim} \cup [\![\ K_1 \Uparrow n \ ]\!]_{prim} \cap [\![[\![\ (K_2 \neg\Uparrow n) \# \Gamma \ ]\!]]_{prim}) \cap ([\![[\![\ \Psi \ ]\!]]_{TESL}^{\geq \ n} \cap [\![[\![\ (K_1 \textit{ implies not } K_2) \# \Phi \ ]\!]]_{TESL}^{\geq \ Suc \ n})\rangle$
   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 ‹$[\![\Gamma, n \vdash ((K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Psi) \triangleright \Phi]\!]_{config}$
$= [\![((K_1 \neg\Uparrow n)\ \#\ \Gamma), n \vdash \Psi \triangleright ((K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Phi)]\!]_{config}$
$\cup [\![((K_1 \Uparrow n)\ \#\ (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)\ \#\ \Gamma), n \vdash \Psi \triangleright ((K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Phi)]\!]_{config}$›
proof −
have *1*:‹$[\![\Gamma, n \vdash (K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Psi \triangleright \Phi]\!]_{config}$
$= [\![[\![\Gamma]\!]]_{prim} \cap [\![[\![(K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Psi]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\![\Phi]\!]]_{TESL}{}^{\geq\ Suc\ n}$›
by *simp*
moreover have ‹$[\![((K_1 \neg\Uparrow n)\ \#\ \Gamma), n \vdash \Psi \triangleright ((K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Phi)]\!]_{config}$
$= [\![[\![(K_1 \neg\Uparrow n)\ \#\ \Gamma]\!]]_{prim} \cap [\![[\![\Psi]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\![(K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Phi]\!]]_{TESL}{}^{\geq\ Suc\ n}$›
by *simp*
moreover have ‹$[\![((K_1 \Uparrow n)\ \#\ (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)\ \#\ \Gamma), n \vdash \Psi \triangleright ((K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Phi)]\!]_{config}$
$= [\![[\![(K_1 \Uparrow n)\ \#\ (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)\ \#\ \Gamma]\!]]_{prim} \cap [\![[\![\Psi]\!]]_{TESL}{}^{\geq\ n}$
$\cap [\![[\![(K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Phi]\!]]_{TESL}{}^{\geq\ Suc\ n}$›
by *simp*
ultimately show *?thesis*
proof −
have ‹$[\![\Gamma, n \vdash (K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Psi \triangleright \Phi]\!]_{config} = [\![[\![\Gamma]\!]]_{prim} \cap ([\![[\![(K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Psi]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\![\Phi]\!]]_{TESL}{}^{\geq\ Suc\ n})$›
using *1* **by** *blast*
then have ‹$[\![\Gamma, n \vdash (K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Psi \triangleright \Phi]\!]_{config} = ([\![K_1 \neg\Uparrow n]\!]_{prim} \cup [\![K_1 \Uparrow n]\!]_{prim} \cap [\![K_2\ @\ n \oplus \delta\tau \Rightarrow K_3]\!]_{prim}) \cap ([\![[\![\Gamma]\!]]_{prim} \cap ([\![[\![\Psi]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\![(K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Phi]\!]]_{TESL}{}^{\geq\ Suc\ n}))$›
using *TESL-interpretation-stepwise-cons-morph TESL-interp-stepwise-timedelayed-coind-unfold*
proof −
have ‹$[\![[\![(K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3)\ \#\ \Psi]\!]]_{TESL}{}^{\geq\ n} = ([\![K_1 \neg\Uparrow n]\!]_{prim} \cup [\![K_1 \Uparrow n]\!]_{prim} \cap [\![K_2\ @\ n \oplus \delta\tau \Rightarrow K_3]\!]_{prim}) \cap [\![K_1\ time{-}delayed\ by\ \delta\tau\ on\ K_2\ implies\ K_3]\!]_{TESL}{}^{\geq\ Suc\ n} \cap [\![[\![\Psi]\!]]_{TESL}{}^{\geq\ n}$›
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 ‹$[\![\Gamma, n \vdash ((K_1\ weakly\ precedes\ K_2)\ \#\ \Psi) \triangleright \Phi]\!]_{config}$

$= [\![(([\#^{\leq} K_2\ n,\ \#^{\leq} K_1\ n] \in (\lambda(x,y).\ x\leq y))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}\rangle$

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

lemma *HeronConf-interp-stepwise-strictly-precedes-cases*:
 shows $\langle[\![\ \Gamma,\ n \vdash ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
 $= [\![(([\#^{\leq} K_2\ n,\ \#^{<} K_1\ n] \in (\lambda(x,y).\ x\leq y))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}\rangle$

proof −
 have $\langle[\![\ \Gamma,\ n \vdash (K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi \triangleright \Phi\ ]\!]_{config} = [\![[\ \Gamma\ ]\!]]_{prim} \cap [\![[\ (K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ \Phi\ ]\!]]_{TESL}{}^{\geq\ Suc\ n}\rangle$
  by *simp*
 moreover have $\langle[\![\ (([\#^{\leq} K_2\ n,\ \#^{<} K_1\ n] \in (\lambda(x,y).\ x\leq y))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
 $= [\![[\ ([\#^{\leq} K_2\ n,\ \#^{<} K_1\ n] \in (\lambda(x,y).\ x\leq y))\ \#\ \Gamma\ ]\!]]_{prim} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} \cap [\![[\ (K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi\ ]\!]]_{TESL}{}^{\geq\ Suc\ n}\rangle$
  by *simp*
 ultimately show *?thesis*
 proof −
  have $\langle[\![\ [\#^{\leq} K_2\ n,\ \#^{<} K_1\ n] \in (\lambda(x,y).\ x\leq y)\ ]\!]_{prim} \cap [\![\ K_1\ \textit{strictly precedes}\ K_2\ ]\!]_{TESL}{}^{\geq\ Suc\ n} \cap [\![[\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n} = [\![[\ (K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi\ ]\!]]_{TESL}{}^{\geq\ n}\rangle$
   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 $\langle[\![\ \Gamma,\ n \vdash ((K_1\ \textit{kills}\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$

$= [\![\; ((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \textit{ kills } K_2) \# \Phi) \;]\!]_{config}$
$\cup [\![\; ((K_1 \Uparrow n) \# (K_2 \neg\Uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \textit{ kills } K_2) \# \Phi) \;]\!]_{config}$›

proof −
have ‹$[\![\; \Gamma, n \vdash ((K_1 \textit{ kills } K_2) \# \Psi) \triangleright \Phi \;]\!]_{config} = [\![[\; \Gamma \;]\!]]_{prim} \cap [\![[\; (K_1 \textit{ kills } K_2) \# \Psi \;]\!]]_{TESL}^{\geq\, n} \cap [\![[\; \Phi \;]\!]]_{TESL}^{\geq\, Suc\, n}$›
by *simp*
moreover have ‹$[\![\; ((K_1 \neg\Uparrow n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \textit{ kills } K_2) \# \Phi) \;]\!]_{config}$
$= [\![[\; (K_1 \neg\Uparrow n) \# \Gamma \;]\!]]_{prim} \cap [\![[\; \Psi \;]\!]]_{TESL}^{\geq\, n} \cap [\![[\; (K_1 \textit{ kills } K_2) \# \Phi \;]\!]]_{TESL}^{\geq\, Suc\, n}$›
by *simp*
moreover have ‹$[\![\; ((K_1 \Uparrow n) \# (K_2 \neg\Uparrow \geq n) \# \Gamma), n \vdash \Psi \triangleright ((K_1 \textit{ kills } K_2) \# \Phi) \;]\!]_{config}$
$= [\![[\; (K_1 \Uparrow n) \# (K_2 \neg\Uparrow \geq n) \# \Gamma \;]\!]]_{prim} \cap [\![[\; \Psi \;]\!]]_{TESL}^{\geq\, n} \cap [\![[\; (K_1 \textit{ kills } K_2) \# \Phi \;]\!]]_{TESL}^{\geq\, Suc\, n}$›
by *simp*
ultimately show *?thesis*
proof −
have ‹$[\![[\; (K_1 \textit{ kills } K_2) \# \Psi \;]\!]]_{TESL}^{\geq\, n} = ([\![\; (K_1 \neg\Uparrow n) \;]\!]_{prim} \cup [\![\; (K_1 \Uparrow n) \;]\!]_{prim} \cap [\![\; (K_2 \neg\Uparrow \geq n) \;]\!]_{prim}) \cap [\![\; (K_1 \textit{ kills } K_2) \;]\!]_{TESL}^{\geq\, Suc\, n} \cap [\![[\; \Psi \;]\!]]_{TESL}^{\geq\, n}$›
using *TESL-interp-stepwise-kills-coind-unfold TESL-interpretation-stepwise-cons-morph*
by *blast*
then show *?thesis*
by *auto*
qed
qed

end

Chapter 7

Main Theorems

theory *Hygge-Theory*
imports
Corecursive-Prop

begin

7.1 Initial configuration

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

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

7.2 Soundness

lemma *sound-reduction*:
assumes ‹$(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$›
shows ‹$[\![[\![\Gamma_1]\!]]\!]_{prim} \cap [\![[\![\Psi_1]\!]]\!]_{TESL}^{\geq n_1} \cap [\![[\![\Phi_1]\!]]\!]_{TESL}^{\geq Suc\ n_1}$
$\supseteq [\![[\![\Gamma_2]\!]]\!]_{prim} \cap [\![[\![\Psi_2]\!]]\!]_{TESL}^{\geq n_2} \cap [\![[\![\Phi_2]\!]]\!]_{TESL}^{\geq Suc\ n_2}$› (**is** *?P*)
proof −
from *assms* **consider**
(a) ‹$(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow_i (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$›
| (b) ‹$(\Gamma_1, n_1 \vdash \Psi_1 \triangleright \Phi_1) \hookrightarrow_e (\Gamma_2, n_2 \vdash \Psi_2 \triangleright \Phi_2)$›
using *operational-semantics-step.simps* **by** *blast*

thus *?thesis*
proof (*cases*)
case *a*
thus *?thesis* **by** (*simp add*: *operational-semantics-intro.simps*)
next
case *b* **thus** *?thesis*
proof (*rule operational-semantics-elim.cases*)
fix $\Gamma\ n\ K_1\ \tau\ K_2\ \Psi\ \Phi$
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 ‹$(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = ((([\tau_{var}\ (K_1,\ n),\ \tau_{var}\ (K_2,\ n)\rfloor \in R)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((\textit{time}-\textit{relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Phi))$›
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 $\langle(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash ((K_1\ \textit{implies not}\ K_2)\ \#\ \Psi) \triangleright \Phi)\rangle$
and $\langle(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((K_1 \Uparrow n)\ \#\ (K_2 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{implies not}\ K_2)\ \#\ \Phi))\rangle$
thus *?P*
using *HeronConf-interp-stepwise-implies-not-cases HeronConf-interpretation.simps*
by *blast*
next
fix $\Gamma\ n\ K_1\ \delta\tau\ K_2\ K_3\ \Psi\ \Phi$
assume $\langle(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash ((K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Psi) \triangleright \Phi)\rangle$
and $\langle(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Phi))\rangle$
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 $\langle(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash ((K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Psi) \triangleright \Phi)\rangle$
and $\langle(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((K_1 \Uparrow n)\ \#\ (K_2\ @\ n \oplus \delta\tau \Rightarrow K_3)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{time}{-}\textit{delayed by}\ \delta\tau\ \textit{on}\ K_2\ \textit{implies}\ K_3)\ \#\ \Phi))\rangle$
thus *?P*
using *HeronConf-interp-stepwise-timedelayed-cases HeronConf-interpretation.simps*
by *blast*
next
fix $\Gamma\ n\ K_1\ K_2\ \Psi\ \Phi$
assume $\langle(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash ((K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Psi) \triangleright \Phi)\rangle$
and $\langle(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((\lceil \#^{\leq}\ K_2\ n,\ \#^{\leq}\ K_1\ n\rceil \in (\lambda(x,\ y).\ x \leq y))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{weakly precedes}\ K_2)\ \#\ \Phi))\rangle$
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 $\langle(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi) \triangleright \Phi)\rangle$
and $\langle(\Gamma_2,\ n_2 \vdash \Psi_2 \triangleright \Phi_2) = (((\lceil \#^{\leq}\ K_2\ n,\ \#^{<}\ K_1\ n\rceil \in (\lambda(x,\ y).\ x \leq y))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi))\rangle$
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 $\langle(\Gamma_1,\ n_1 \vdash \Psi_1 \triangleright \Phi_1) = (\Gamma,\ n \vdash ((K_1\ \textit{kills}\ K_2)\ \#\ \Psi) \triangleright \Phi)\rangle$

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*

7.3 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*

7.4 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*: ‹$[\![\ \Gamma,\ n \vdash ((K_1\ sporadic\ \tau\ on\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
$= [\![\ \Gamma,\ n \vdash \Psi \triangleright ((K_1\ sporadic\ \tau\ on\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$\cup\ [\![\ ((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config}$›
using *HeronConf-interp-stepwise-sporadicon-cases* **by** *simp*
have *br1*: ‹$\varrho \in [\![\ \Gamma,\ n \vdash \Psi \triangleright ((K_1\ sporadic\ \tau\ on\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$\Longrightarrow \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$

$((\Gamma,\ n \vdash ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Psi) \triangleright \Phi) \hookrightarrow^{k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\ \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}\rangle$

proof −
assume *h1*: $\langle\varrho \in [\![\ \Gamma,\ n \vdash \Psi \triangleright ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Phi)\ ]\!]_{config}\rangle$
hence $\langle\exists\,\Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((\Gamma,\ n \vdash \Psi \triangleright ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Phi))$
$\hookrightarrow^{k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ (\varrho \in [\![\ \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config})\rangle$
using *h1 SporadicOn.prems* **by** *simp*
from *this* **obtain** $\Gamma_k\ \Psi_k\ \Phi_k\ k$ **where**
fp:$\langle((\Gamma,\ n \vdash \Psi \triangleright ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Phi)) \hookrightarrow^{k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\ \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}\rangle$ **by** *blast*
have
$\langle(\Gamma,\ n \vdash ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Psi) \triangleright \Phi) \hookrightarrow (\Gamma,\ n \vdash \Psi \triangleright ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Phi))\rangle$
by (*simp add*: *elims-part sporadic-on-e1*)
with *fp relpowp-Suc-I2* **have**
$\langle((\Gamma,\ n \vdash ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Psi) \triangleright \Phi) \hookrightarrow^{\mathit{Suc}\ k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))\rangle$ **by** *auto*
thus *?thesis* **using** *fp* **by** *blast*
qed
have *br2*: $\langle\varrho \in [\![\ ((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config}$
$\Longrightarrow \exists\,\Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((\Gamma,\ n \vdash ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow^{k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\ \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}\rangle$
proof −
assume *h2*: $\langle\varrho \in [\![\ ((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config}\rangle$
hence $\langle\exists\,\Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi)$
$\hookrightarrow^{k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\ \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}\rangle$
using *h2 SporadicOn.prems* **by** *simp*

from *this* **obtain** $\Gamma_k\ \Psi_k\ \Phi_k\ k$ **where** *fp*:$\langle((((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi)$
$\hookrightarrow^{k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))\rangle$
and *rc*:$\langle\varrho \in [\![\ \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}\rangle$ **by** *blast*
have *pc*:$\langle(\Gamma,\ n \vdash ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow (((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi)\rangle$ **by** (*simp add*: *elims-part sporadic-on-e2*)
hence $\langle(\Gamma,\ n \vdash (K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Psi \triangleright \Phi) \hookrightarrow^{\mathit{Suc}\ k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k)\rangle$
using *fp relpowp-Suc-I2* **by** *auto*
with *rc* **show** *?thesis* **by** *blast*
qed
from *branches SporadicOn.prems*(*2*) **have**
$\langle\varrho \in [\![\ \Gamma,\ n \vdash \Psi \triangleright ((K_1\ \mathit{sporadic}\ \tau\ \mathit{on}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$\cup\ [\![\ ((K_1 \Uparrow n)\ \#\ (K_2 \Downarrow n\ @\ \tau)\ \#\ \Gamma),\ n \vdash \Psi \triangleright \Phi\ ]\!]_{config}\rangle$
by *simp*

with *br1 br2* **show** *?case* **by** *blast*
next
case (*TagRelation* K_1 K_2 *R*)
have *branches*: $\langle [\![\, \Gamma,\ n \vdash ((\mathit{time{-}relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Psi) \triangleright \Phi \,]\!]_{config}$
$= [\![\, ((\lfloor \tau_{var}(K_1,\ n),\ \tau_{var}(K_2,\ n) \rfloor \in R)\ \#\ \Gamma),\ n$
$\vdash \Psi \triangleright ((\mathit{time{-}relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Phi) \,]\!]_{config} \rangle$
using *HeronConf-interp-stepwise-tagrel-cases* **by** *simp*
thus *?case*
proof −
have $\langle \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$
$(((((\lfloor \tau_{var}(K_1,\ n),\ \tau_{var}(K_2,\ n) \rfloor \in R)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((\mathit{time{-}relation}$ $\lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Phi))$
$\hookrightarrow^{k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k)) \wedge \varrho \in [\![\, \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k \,]\!]_{config} \rangle$
using *TagRelation.prems* **by** *simp*

from *this* **obtain** Γ_k Ψ_k Φ_k k
where *fp*: $\langle (((((\lfloor \tau_{var}(K_1,\ n),\ \tau_{var}(K_2,\ n) \rfloor \in R)\ \#\ \Gamma),\ n$
$\vdash \Psi \triangleright ((\mathit{time{-}relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Phi))$
$\hookrightarrow^{k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))) \rangle$
and *rc*: $\langle \varrho \in [\![\, \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k \,]\!]_{config} \rangle$ **by** *blast*
have *pc*: $\langle (\Gamma,\ n \vdash ((\mathit{time{-}relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow (((\lfloor \tau_{var}\ (K_1,\ n),\ \tau_{var}\ (K_2,\ n) \rfloor \in R)\ \#\ \Gamma),\ n$
$\vdash \Psi \triangleright ((\mathit{time{-}relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Phi)) \rangle$
by (*simp add*: *elims-part tagrel-e*)
hence $\langle (\Gamma,\ n \vdash (\mathit{time{-}relation}\ \lfloor K_1,\ K_2 \rfloor \in R)\ \#\ \Psi \triangleright \Phi) \hookrightarrow^{\mathit{Suc}\ k} (\Gamma_k,\ \mathit{Suc}$ $n \vdash \Psi_k \triangleright \Phi_k) \rangle$
using *fp relpowp-Suc-I2* **by** *auto*
with *rc* **show** *?thesis* **by** *blast*
qed
next
case (*Implies* K_1 K_2)
have *branches*: $\langle [\![\, \Gamma,\ n \vdash ((K_1\ \mathit{implies}\ K_2)\ \#\ \Psi) \triangleright \Phi \,]\!]_{config}$
$= [\![\, ((K_1\ \neg\Uparrow\ n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2)\ \#\ \Phi) \,]\!]_{config}$
$\cup [\![\, ((K_1 \Uparrow n)\ \#\ (K_2 \Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2)\ \#\ \Phi) \,]\!]_{config} \rangle$
using *HeronConf-interp-stepwise-implies-cases* **by** *simp*
moreover have *br1*: $\langle \varrho \in [\![\, ((K_1\ \neg\Uparrow\ n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2)\ \#$ $\Phi) \,]\!]_{config}$
$\Longrightarrow \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((\Gamma,\ n \vdash ((K_1\ \mathit{implies}\ K_2)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow^{k} (\Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\, \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k \,]\!]_{config} \rangle$
proof −
assume *h1*: $\langle \varrho \in [\![\, ((K_1\ \neg\Uparrow\ n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2)\ \#\ \Phi)$ $]\!]_{config} \rangle$
then have $\langle \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$
$((((K_1\ \neg\Uparrow\ n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \mathit{implies}\ K_2)\ \#\ \Phi)) \hookrightarrow^{k} (\Gamma_k,$ $\mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ \varrho \in [\![\, \Gamma_k,\ \mathit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k \,]\!]_{config} \rangle$
using *h1 Implies.prems* **by** *simp*
from *this* **obtain** Γ_k Ψ_k Φ_k k **where**

fp:‹((((K_1 $\neg\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies* K_2) # Φ)) $\hookrightarrow^k$ (Γ_k, *Suc* n $\vdash \Psi_k \triangleright \Phi_k$)))›
and *rc*:‹$\varrho \in [\![\Gamma_k$, *Suc* $n \vdash \Psi_k \triangleright \Phi_k]\!]_{config}$› **by** *blast*
have *pc*:‹(Γ, $n \vdash$ (K_1 *implies* K_2) # $\Psi \triangleright \Phi$)
$\hookrightarrow$ (((K_1 $\neg\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies* K_2) # Φ))›
by (*simp add*: *elims-part implies-e1*)
hence ‹(Γ, $n \vdash$ (K_1 *implies* K_2) # $\Psi \triangleright \Phi$) $\hookrightarrow^{Suc\ k}$ (Γ_k, *Suc* $n \vdash \Psi_k \triangleright \Phi_k$)›
using *fp relpowp-Suc-I2* **by** *auto*
with *rc* **show** *?thesis* **by** *blast*
qed
moreover have *br2*: ‹$\varrho \in [\![$ ((K_1 $\Uparrow$ n) # (K_2 $\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies* K_2) # Φ) $]\!]_{config}$
$\Longrightarrow \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$ ((Γ, $n \vdash$ ((K_1 *implies* K_2) # Ψ) $\triangleright$ Φ)
$\hookrightarrow^k$ (Γ_k, *Suc* $n \vdash \Psi_k \triangleright \Phi_k$))
$\wedge\ \varrho \in [\![\Gamma_k$, *Suc* $n \vdash \Psi_k \triangleright \Phi_k]\!]_{config}$›
proof −
assume *h2*: ‹$\varrho \in [\![$ ((K_1 $\Uparrow$ n) # (K_2 $\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies* K_2) # Φ) $]\!]_{config}$›
then have ‹$\exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$ (
(((K_1 $\Uparrow$ n) # (K_2 $\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies* K_2) # Φ)) $\hookrightarrow^k$ (Γ_k, *Suc* $n \vdash \Psi_k \triangleright \Phi_k$)
) $\wedge\ \varrho \in [\![\Gamma_k$, *Suc* $n \vdash \Psi_k \triangleright \Phi_k]\!]_{config}$›
using *h2 Implies.prems* **by** *simp*
from *this* **obtain** $\Gamma_k\ \Psi_k\ \Phi_k\ k$ **where**
fp:‹(((K_1 $\Uparrow$ n) # (K_2 $\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies* K_2) # Φ))
$\hookrightarrow^k$ (Γ_k, *Suc* $n \vdash \Psi_k \triangleright \Phi_k$)›
and *rc*:‹$\varrho \in [\![\Gamma_k$, *Suc* $n \vdash \Psi_k \triangleright \Phi_k]\!]_{config}$› **by** *blast*
have ‹(Γ, $n \vdash$ ((K_1 *implies* K_2) # Ψ) $\triangleright$ Φ)
$\hookrightarrow$ (((K_1 $\Uparrow$ n) # (K_2 $\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies* K_2) # Φ))›
by (*simp add*: *elims-part implies-e2*)
hence ‹(Γ, $n \vdash$ ((K_1 *implies* K_2) # Ψ) $\triangleright$ Φ) $\hookrightarrow^{Suc\ k}$ (Γ_k, *Suc* $n \vdash \Psi_k \triangleright \Phi_k$)›
using *fp relpowp-Suc-I2* **by** *auto*
with *rc* **show** *?thesis* **by** *blast*
qed
ultimately show *?case* **using** *Implies.prems*(*2*) **by** *blast*
next
case (*ImpliesNot* K_1 K_2)
have *branches*: ‹$[\![$ Γ, $n \vdash$ ((K_1 *implies not* K_2) # Ψ) $\triangleright$ Φ $]\!]_{config}$
= $[\![$ ((K_1 $\neg\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies not* K_2) # Φ) $]\!]_{config}$
∪ $[\![$ ((K_1 $\Uparrow$ n) # (K_2 $\neg\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies not* K_2) # Φ) $]\!]_{config}$›
using *HeronConf-interp-stepwise-implies-not-cases* **by** *simp*
moreover have *br1*: ‹$\varrho \in [\![$ ((K_1 $\neg\Uparrow$ n) # Γ), $n \vdash \Psi \triangleright$ ((K_1 *implies not* K_2) # Φ) $]\!]_{config}$
$\Longrightarrow \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.$ ((Γ, $n \vdash$ ((K_1 *implies not* K_2) # Ψ) $\triangleright$ Φ)
$\hookrightarrow^k$ (Γ_k, *Suc* $n \vdash \Psi_k \triangleright \Phi_k$))
$\wedge\ \varrho \in [\![\Gamma_k$, *Suc* $n \vdash \Psi_k \triangleright \Phi_k]\!]_{config}$›
proof −

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

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

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

by *auto*
with *rc* **show** *?thesis* **by** *blast*
qed
ultimately show *?case* **using** *WeaklyPrecedes.prems(2)* **by** *blast*
next
case (*StrictlyPrecedes* K_1 K_2)
have ⟨$[\![\, \Gamma,\ n \vdash ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config} =$
$[\![\ ((\lceil \#^{\leq}\ K_2\ n,\ \#^{<}\ K_1\ n \rceil \in (\lambda(x,\ y).\ x \leq y))\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{strictly}$ $\textit{precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$⟩
using *HeronConf-interp-stepwise-strictly-precedes-cases* **by** *simp*
moreover have ⟨$\varrho \in [\![\ ((\lceil \#^{\leq}\ K_2\ n,\ \#^{<}\ K_1\ n \rceil \in (\lambda(x,\ y).\ x \leq y))\ \#\ \Gamma),\ n$
$\vdash \Psi \triangleright ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$\Longrightarrow (\exists \Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((\Gamma,\ n \vdash ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow^{k} (\Gamma_k,\ \textit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$
$\wedge\ (\varrho \in [\![\ \Gamma_k,\ \textit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}))$⟩
proof –
assume ⟨$\varrho \in [\![\ ((\lceil \#^{\leq}\ K_2\ n,\ \#^{<}\ K_1\ n \rceil \in (\lambda(x,\ y).\ x \leq y))\ \#\ \Gamma),\ n$
$\vdash \Psi \triangleright ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$⟩
hence ⟨$\exists \Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((((\lceil \#^{\leq}\ K_2\ n,\ \#^{<}\ K_1\ n \rceil \in (\lambda(x,\ y).\ x \leq y))\ \#\ \Gamma),$
n
$\vdash \Psi \triangleright ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi))$
$\hookrightarrow^{k} (\Gamma_k,\ \textit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))\ \wedge\ (\varrho \in [\![\ \Gamma_k,\ \textit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k$
$]\!]_{config})$⟩
using *StrictlyPrecedes.prems* **by** *simp*
from *this* **obtain** Γ_k Ψ_k Φ_k k
where *fp*:⟨$(((\lceil \#^{\leq}\ K_2\ n,\ \#^{<}\ K_1\ n \rceil \in (\lambda(x,\ y).\ x \leq y))\ \#\ \Gamma),\ n$
$\vdash \Psi \triangleright ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi))$
$\hookrightarrow^{k} (\Gamma_k,\ \textit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k)$⟩
and *rc*:⟨$\varrho \in [\![\ \Gamma_k,\ \textit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k\ ]\!]_{config}$⟩ **by** *blast*
have ⟨$(\Gamma,\ n \vdash ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow (((\lceil \#^{\leq}\ K_2\ n,\ \#^{<}\ K_1\ n \rceil \in (\lambda(x,\ y).\ x \leq y))\ \#\ \Gamma),\ n$
$\vdash \Psi \triangleright ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Phi))$⟩ **by** (*simp add*: *elims-part* *strictly-precedes-e*)
with *fp relpowp-Suc-I2* **have** ⟨$(\Gamma,\ n \vdash ((K_1\ \textit{strictly precedes}\ K_2)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow^{\textit{Suc}\ k} (\Gamma_k,\ \textit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k)$⟩
by *auto*
with *rc* **show** *?thesis* **by** *blast*
qed
ultimately show *?case* **using** *StrictlyPrecedes.prems(2)* **by** *blast*
next
case (*Kills* K_1 K_2)
have *branches*: ⟨$[\![\ \Gamma,\ n \vdash ((K_1\ \textit{kills}\ K_2)\ \#\ \Psi) \triangleright \Phi\ ]\!]_{config}$
$= [\![\ ((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{kills}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$
$\cup\ [\![\ ((K_1 \Uparrow n)\ \#\ (K_2 \neg\Uparrow \geq n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{kills}\ K_2)\ \#\ \Phi)\ ]\!]_{config}$⟩
using *HeronConf-interp-stepwise-kills-cases* **by** *simp*
moreover have *br1*: ⟨$\varrho \in [\![\ ((K_1 \neg\Uparrow n)\ \#\ \Gamma),\ n \vdash \Psi \triangleright ((K_1\ \textit{kills}\ K_2)\ \#\ \Phi)$
$]\!]_{config}$
$\Longrightarrow \exists \Gamma_k\ \Psi_k\ \Phi_k\ k.\ ((\Gamma,\ n \vdash ((K_1\ \textit{kills}\ K_2)\ \#\ \Psi) \triangleright \Phi)$
$\hookrightarrow^{k} (\Gamma_k,\ \textit{Suc}\ n \vdash \Psi_k \triangleright \Phi_k))$

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

lemma *instant-index-increase-generalized*:
assumes $\langle n < n_k\rangle$

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

7.5 Local termination

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

fun *measure-interpretation-config* :: ‹$'\tau$ :: *linordered-field config* $\Rightarrow$ *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** $\Gamma_1\ n_1\ \Psi_1\ \Phi_1$ **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** $\Gamma_2\ n_2\ \Psi_2\ \Phi_2$ **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* $(\lambda(\mathcal{S}_1{::}\ 'a :: \textit{linordered-field config})\ \mathcal{S}_2.\ (\mathcal{S}_1\ \hookrightarrow_e{}^{\leftarrow}\ \mathcal{S}_2))$›
proof (*simp add*: *wfP-def*)
show ‹*wf* $\{((\mathcal{S}_1{::}\ 'a :: \textit{linordered-field config}),\ \mathcal{S}_2).\ \mathcal{S}_1 \hookrightarrow_e{}^{\leftarrow} \mathcal{S}_2\}$›
proof (*rule wf-subset*)
have ‹*measure* $\mu_{config} = \{\ (\mathcal{S}_2,\ (\mathcal{S}_1{::}\ 'a :: \textit{linordered-field config})).\ \mu_{config}$
$\mathcal{S}_2 < \mu_{config}\ \mathcal{S}_1\ \}$›
by (*simp add*: *inv-image-def less-eq measure-def*)
thus ‹$\{((\mathcal{S}_1{::}\ 'a :: \textit{linordered-field config}),\ \mathcal{S}_2).\ \mathcal{S}_1 \hookrightarrow_e{}^{\leftarrow} \mathcal{S}_2\} \subseteq$ (*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 8

Properties of TESL

8.1 Stuttering Invariance

theory *StutteringDefs*

imports *Denotational*

begin

8.1.1 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

8.1.2 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

8.1.3 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 ≤ k ∧ k < n ∧ hamlet ((Rep-run r) (f k) c)}›
  (is ‹?IMG = image f ?SET›)
proof
  { fix k assume h:‹k ∈ ?IMG›
    from h obtain k0 where k0prop:‹f k0 = k ∧ hamlet ((Rep-run r) (f k0) c)›
      using ticks-image[OF assms] by blast
    with h have ‹k ∈ image f ?SET›
      using assms dilating-fun-def strict-mono-less strict-mono-less-eq by fastforce
  } thus ‹?IMG ⊆ image f ?SET› ..
next
  { fix k assume h:‹k ∈ image f ?SET›
    from h obtain k0 where k0prop:‹k = f k0 ∧ k0 ∈ ?SET› by blast
    hence ‹k ∈ ?IMG›
      using assms dilating-fun-def strict-mono-less strict-mono-less-eq by fastforce
  } thus ‹image f ?SET ⊆ ?IMG› ..
qed

lemma dilating-fun-image-right:
  assumes ‹dilating-fun f r›
  shows   ‹{k. f m < k ∧ k ≤ f n ∧ hamlet ((Rep-run r) k c)}
      = image f {k. m < k ∧ k ≤ n ∧ hamlet ((Rep-run r) (f k) c)}›
  (is ‹?IMG = image f ?SET›)
proof
  { fix k assume h:‹k ∈ ?IMG›
    from h obtain k0 where k0prop:‹f k0 = k ∧ hamlet ((Rep-run r) (f k0) c)›
      using ticks-image[OF assms] by blast
    with h have ‹k ∈ image f ?SET›
      using assms dilating-fun-def strict-mono-less strict-mono-less-eq by fastforce
  } thus ‹?IMG ⊆ image f ?SET› ..
next
  { fix k assume h:‹k ∈ image f ?SET›
    from h obtain k0 where k0prop:‹k = f k0 ∧ k0 ∈ ?SET› by blast
    hence ‹k ∈ ?IMG›
      using assms dilating-fun-def strict-mono-less strict-mono-less-eq by fastforce
  } thus ‹image f ?SET ⊆ ?IMG› ..
qed

lemma dilating-fun-image:
  assumes ‹dilating-fun f r›
  shows   ‹{k. f m ≤ k ∧ k ≤ f n ∧ hamlet ((Rep-run r) k c)}
      = image f {k. m ≤ k ∧ k ≤ n ∧ hamlet ((Rep-run r) (f k) c)}›
  (is ‹?IMG = image f ?SET›)
proof
  { fix k assume h:‹k ∈ ?IMG›
    from h obtain k0 where k0prop:‹f k0 = k ∧ hamlet ((Rep-run r) (f k0) c)›
      using ticks-image[OF assms] by blast
    with h have ‹k ∈ image f ?SET›
      using assms dilating-fun-def strict-mono-less-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 ‹$\exists\, k_0.\ f\ k_0 = k \wedge time\ ((Rep\text{-}run\ sub)\ k_0\ c) = \tau$›
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** ‹$time\ ((Rep\text{-}run\ sub)\ k_0\ c) = \tau$› **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 ‹$hamlet\ ((Rep\text{-}run\ sub)\ n\ a) = hamlet\ ((Rep\text{-}run\ r)\ (f\ n)\ a)$›
using *assms* **by** (*simp add*: *dilating-def*)

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

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

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

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 ‹$\{k.\ hamlet\ ((Rep\text{-}run\ r)\ k\ c)\} = image\ f\ \{k.\ hamlet\ ((Rep\text{-}run\ sub)\ k\ c)\}$›
(**is** ‹$?R = image\ f\ ?S$›)
proof
{ fix k **assume** h:‹$k \in\ ?R$›
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** ‹$hamlet\ ((Rep\text{-}run\ sub)\ k_0\ c)$› **by** *blast*
with *k0prop* **have** ‹$k \in image\ f\ ?S$› **by** *blast*
}
thus ‹$?R \subseteq image\ f\ ?S$› **by** *blast*
next
{ fix k **assume** h:‹$k \in image\ f\ ?S$›
from *this* **obtain** k_0 **where** ‹$f\ k_0 = k \wedge hamlet\ ((Rep\text{-}run\ sub)\ k_0\ c)$› **by** *blast*
with *assms* **have** ‹$k \in\ ?R$› **using** *ticks-sub* **by** *blast*
}
thus ‹$image\ f\ ?S \subseteq\ ?R$› **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 ‹$\exists\, k_0.\ f\, k_0 = k \wedge$ *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* ∧ *i* < *f n* ∧ *hamlet* ((*Rep-run r*) *i c*)}
= *image f* {*i*. *m* < *i* ∧ *i* < *n* ∧ *hamlet* ((*Rep-run sub*) *i c*)}›
(**is** ‹*?RUN* = *image f ?SUB*›)
proof
{ **fix** *i* **assume** *h*:‹*i* ∈ *?SUB*›
hence ‹*m* < *i* ∧ *i* < *n*› **by** *simp*
hence ‹*f m* < *f i* ∧ *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* ∈ *?RUN*› **by** *simp*
} **thus** ‹*image f ?SUB* ⊆ *?RUN*› **by** *blast*
next
{ **fix** *i* **assume** *h*:‹*i* ∈ *?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* ∧ *hamlet* ((*Rep-run sub*) i_0 *c*)› **by** *blast*
with *h* **have** ‹*f m* < *f* i_0 ∧ *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 ∧ i_0 < *n*› **using** *strict-mono-less strict-mono-less-eq* **by** *blast*
with *i0prop* **have** ‹∃ i_0. *f* i_0 = *i* ∧ i_0 ∈ *?SUB*› **by** *blast*
} **thus** ‹*?RUN* ⊆ *image f ?SUB*› **by** *blast*
qed

lemma *dilated-ticks-left*:
assumes ‹*dilating f sub r*›
shows ‹{*i*. *f m* ≤ *i* ∧ *i* < *f n* ∧ *hamlet* ((*Rep-run r*) *i c*)}
= *image f* {*i*. *m* ≤ *i* ∧ *i* < *n* ∧ *hamlet* ((*Rep-run sub*) *i c*)}›
(**is** ‹*?RUN* = *image f ?SUB*›)
proof
{ **fix** *i* **assume** *h*:‹*i* ∈ *?SUB*›
hence ‹*m* ≤ *i* ∧ *i* < *n*› **by** *simp*
hence ‹*f m* ≤ *f i* ∧ *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* ∈ *?RUN*› **by** *simp*
} **thus** ‹*image f ?SUB* ⊆ *?RUN*› **by** *blast*
next

```
  { fix i assume h:‹i ∈ ?RUN›
    hence ‹hamlet ((Rep-run r) i c)› by simp
    from ticks-imp-ticks-subk[OF assms this]
      obtain i0 where i0prop:‹f i0 = i ∧ hamlet ((Rep-run sub) i0 c)› by blast
    with h have ‹f m ≤ f i0 ∧ f i0 < f n› by simp
    moreover have ‹strict-mono f› using assms dilating-def dilating-fun-def by blast
    ultimately have ‹m ≤ i0 ∧ i0 < n› using strict-mono-less strict-mono-less-eq by blast
    with i0prop have ‹∃ i0. f i0 = i ∧ i0 ∈ ?SUB› by blast
  } thus ‹?RUN ⊆ image f ?SUB› by blast
qed

lemma dilated-ticks-right:
  assumes ‹dilating f sub r›
  shows   ‹{i. f m < i ∧ i ≤ f n ∧ hamlet ((Rep-run r) i c)}
          = image f {i. m < i ∧ i ≤ n ∧ hamlet ((Rep-run sub) i c)}›
    (is ‹?RUN = image f ?SUB›)
proof
  { fix i  assume h:‹i ∈ ?SUB›
    hence ‹m < i ∧ i ≤ n› by simp
    hence ‹f m < f i ∧ 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 ∈ ?RUN› by simp
  } thus ‹image f ?SUB ⊆ ?RUN› by blast
next
  { fix i assume h:‹i ∈ ?RUN›
    hence ‹hamlet ((Rep-run r) i c)› by simp
    from ticks-imp-ticks-subk[OF assms this]
      obtain i0 where i0prop:‹f i0 = i ∧ hamlet ((Rep-run sub) i0 c)› by blast
    with h have ‹f m < f i0 ∧ f i0 ≤ f n› by simp
    moreover have ‹strict-mono f› using assms dilating-def dilating-fun-def by blast
    ultimately have ‹m < i0 ∧ i0 ≤ n› using strict-mono-less strict-mono-less-eq by blast
    with i0prop have ‹∃ i0. f i0 = i ∧ i0 ∈ ?SUB› by blast
  } thus ‹?RUN ⊆ image f ?SUB› by blast
qed

lemma dilated-ticks:
  assumes ‹dilating f sub r›
  shows   ‹{i. f m ≤ i ∧ i ≤ f n ∧ hamlet ((Rep-run r) i c)}
          = image f {i. m ≤ i ∧ i ≤ n ∧ hamlet ((Rep-run sub) i c)}›
    (is ‹?RUN = image f ?SUB›)
proof
  { fix i assume h:‹i ∈ ?SUB›
    hence ‹m ≤ i ∧ i ≤ 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 ‹$\bigwedge$*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 ‹$\neg hamlet\ ((Rep\text{-}run\ r)\ k\ c)$›
proof −
  from *assms(1)* **have** *smf*:‹*strict-mono f*› **by** (*simp add*: *dilating-def dilating-fun-def*)
  { fix *k* **assume** *h*:‹$f\ n < k \wedge k < f\ (Suc\ n) \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)$›
    hence ‹$\exists\, k_0.\ f\ k_0 = k$› **using** *assms(1) dilating-def dilating-fun-def* **by** *blast*
    from *this* **obtain** k_0 **where** ‹$f\ k_0 = k$› **by** *blast*
    with *h* **have** ‹$f\ n < f\ k_0 \wedge f\ k_0 < f\ (Suc\ n)$› **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 ‹$tick\text{-}count\ r\ c\ (f\ (Suc\ n)) = tick\text{-}count\ r\ c\ (f\ n) + card\ \{k.\ k = f\ (Suc\ n) \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)\}$›
proof −
  have *smf*:‹*strict-mono f*› **using** *assms dilating-def dilating-fun-def* **by** *blast*
  moreover have ‹$finite\ \{k.\ k \leq f\ n \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)\}$› **by** *simp*
  moreover have $*$:‹$finite\ \{k.\ f\ n < k \wedge k \leq f\ (Suc\ n) \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)\}$› **by** *simp*
  ultimately have ‹$\{k.\ k \leq f\ (Suc\ n) \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)\} =$
      $\{k.\ k \leq f\ n \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)\}$
      $\cup \{k.\ f\ n < k \wedge k \leq f\ (Suc\ n) \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)\}$›
    by (*simp add*: *nat-interval-union strict-mono-less-eq*)
  moreover have ‹$\{k.\ k \leq f\ n \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)\}$
      $\cap \{k.\ f\ n < k \wedge k \leq f\ (Suc\ n) \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)\} = \{\}$›
    by *auto*
  ultimately have ‹$card\ \{k.\ k \leq f\ (Suc\ n) \wedge hamlet\ (Rep\text{-}run\ r\ k\ c)\} =$
      $card\ \{k.\ k \leq f\ n \wedge hamlet\ (Rep\text{-}run\ r\ k\ c)\}$
      $+\ card\ \{k.\ f\ n < k \wedge k \leq f\ (Suc\ n) \wedge hamlet\ (Rep\text{-}run\ r\ k\ c)\}$›
    by (*simp add*: $*$ *card-Un-disjoint*)
  moreover from *no-tick-before-suc*[*OF assms*] **have**
    ‹$\{k.\ f\ n < k \wedge k \leq f\ (Suc\ n) \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)\} =$
      $\{k.\ k = f\ (Suc\ n) \wedge hamlet\ ((Rep\text{-}run\ r)\ k\ c)\}$›
    using *smf strict-mono-less* **by** *fastforce*
  ultimately show *?thesis* **by** (*simp add*: *tick-count-def*)
qed

lemma *card-sing-prop*:‹*card* $\{i.\ i = n \wedge P\ i\}$ = (*if P n then 1 else 0*)›
proof (*cases* ‹*P n*›)
  case *True*
    hence ‹$\{i.\ i = n \wedge P\ i\} = \{n\}$› **by** (*simp add*: *Collect-conv-if*)
    with ‹*P n*› **show** *?thesis* **by** *simp*
next
  case *False*
    hence ‹$\{i.\ i = n \wedge P\ i\} = \{\}$› **by** (*simp add*: *Collect-conv-if*)
    with ‹$\neg P\ n$› **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*)› ‹λk. *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* $\{i.\ i \leq n \wedge$ *hamlet* ((*Rep-run sub*) *i c*)$\}$›
    using *tick-count-def*[*of* ‹*sub*› ‹*c*› ‹*n*›] **.**
  also have ‹... = *card* (*image f* $\{i.\ i \leq n \wedge$ *hamlet* ((*Rep-run sub*) *i c*)$\}$)›
    using *assms dilating-def dilating-injects*[*OF assms*] **by** (*simp add*: *card-image*)
  also have ‹... = *card* $\{i.\ i \leq f\ n \wedge$ *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*)$\}$
$\cup$ $\{i.\ f\, n_p < i \wedge i \leq n \wedge$ *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*)$\}$
$\cap$ $\{i.\ f\, n_p < i \wedge i \leq n \wedge$ *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 ‹$(f\, n) < k \wedge k < (f\ (Suc\ n))$›
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 ‹$\forall i.\ k \leq i \longrightarrow f\, n < i$› **by** *simp*
with *no-tick-before-suc*[*OF assms*(*1*)] **have**
$*$:‹$\forall i.\ k \leq i \wedge i < f\ (Suc\ n) \longrightarrow \neg$*hamlet* ((*Rep-run r*) *i c*)› **by** *blast*
from *tick-count-strict-def* **have** ‹*tick-count-strict r c (f (Suc n))* = *card* $\{i.\ i < f\ (Suc\ n) \wedge$ *hamlet* ((*Rep-run r*) *i c*)$\}$› **.**
also have ‹*...* = *card* $\{i.\ i < k \wedge$ *hamlet* ((*Rep-run r*) *i c*)$\}$ + *card* $\{i.\ k \leq i \wedge i < f\ (Suc\ n) \wedge$ *hamlet* ((*Rep-run r*) *i c*)$\}$›
using *card-mnm'* *assms*(*2*) **by** *simp*

also have ‹... = *card* $\{i.\ i < k \land$ *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* $\{i.\ i < n \land$ *hamlet* ((*Rep-run sub*) *i c*)$\}$›
using *tick-count-strict-def*[*of* ‹*sub*› ‹*c*› ‹*n*›] **.**
also have ‹... = *card* (*image f* $\{i.\ i < n \land$ *hamlet* ((*Rep-run sub*) *i c*)$\}$)›
using *assms dilating-def dilating-injects*[*OF assms*] **by** (*simp add*: *card-image*)
also have ‹... = *card* $\{i.\ i < f\ n \land$ *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 ‹$m \leq n$›
shows ‹*card* $\{i{::}nat.\ i \leq m \land P\ i\} \leq$ *card* $\{i.\ i \leq n \land P\ i\}$›
proof −
from *assms* **have** ‹$\{i.\ i \leq m \land P\ i\} \subseteq \{i.\ i \leq n \land P\ i\}$› **by** *auto*
moreover have ‹*finite* $\{i.\ i \leq n \land P\ i\}$› **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 ‹$x \leq y$›
from *card-prop-mono*[*OF this*] **have** ‹*tick-count r c x* $\leq$ *tick-count r c y*›
unfolding *tick-count-def* **by** *simp*
} thus ‹$\bigwedge x\ y.\ x \leq y \Longrightarrow$ *tick-count r c x* $\leq$ *tick-count r c y*› **.**
qed

lemma *greatest-prev-image*:
assumes ‹*dilating f sub r*›
shows ‹$(\nexists n_0.\ f\ n_0 = n) \Longrightarrow (\exists n_p.\ f\ n_p < n \land (\forall k.\ f\ n_p < k \land k \leq n \longrightarrow (\nexists k_0.\ f\ k_0 = k)))$›
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-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-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*)› ‹λ*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*:‹∀ *n*. (*dil-inverse f*) *n* ≤ *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*:‹∀ *n c k*. *f* ((*dil-inverse f*) *n*) < *k* ∧ *k* ≤ *n*
⟶ ¬ *hamlet* ((*Rep-run r*) *k c*)›
using *not-image-stut*[*OF assms*] *no-img-tick* **unfolding** *dil-inverse-def* **by** *blast*

have *5*:‹(∀ *n c k*. *f* ((*dil-inverse f*) *n*) ≤ *k* ∧ *k* ≤ *n*
⟶ *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*) ≤ *k* ∧ *k* ≤ *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* ≤ *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 ‹∀ *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* ≤ *n*}) < *k* ∧ *k* ≤ *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

8.1.4 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 ‹$\forall n.$ *hamlet* ((*Rep-run* r) n c_1) $\longrightarrow \neg$ *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* $\ll$ r›
and ‹*sub* $\in [\![c_1$ *weakly precedes* $c_2]\!]_{TESL}$›
shows ‹$r \in [\![c_1$ *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* $\in \{r. \forall n.$ (*run-tick-count* r c_2 n) $\leq$ (*run-tick-count* r c_1 n)$\}$› **by** *simp*
hence ‹$\forall n.$ (*run-tick-count sub* c_2 n) $\leq$ (*run-tick-count sub* c_1 n)› **by** *simp*
from *dil-tick-count*[*OF assms*(*1*) *this*] **have** ‹$\forall n.$ (*run-tick-count* r c_2 n) $\leq$ (*run-tick-count* r c_1 n)› **by** *simp*
thus *?thesis* **by** *simp*
qed

theorem *strictly-precedes-sub*:
assumes ‹*sub* $\ll$ r›
and ‹*sub* $\in [\![c_1$ *strictly precedes* $c_2]\!]_{TESL}$›
shows ‹$r \in [\![c_1$ *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* $\in \{ \varrho. \forall n{::}nat.$ (*run-tick-count* ϱ c_2 n) $\leq$ (*run-tick-count-strictly* ϱ c_1 n) $\}$› **by** *simp*
with *strictly-precedes-alt-def2*[*of* ‹c_2› ‹c_1›] **have**
‹*sub* $\in \{ \varrho.$ ($\neg$*hamlet* ((*Rep-run* ϱ) *0* c_2)) $\wedge$ ($\forall n{::}nat.$ (*run-tick-count* ϱ c_2 (*Suc* n)) $\leq$ (*run-tick-count* ϱ c_1 n)) $\}$›
by *blast*
hence ‹($\neg$*hamlet* ((*Rep-run sub*) *0* c_2)) $\wedge$ ($\forall n{::}nat.$ (*run-tick-count sub* c_2 (*Suc* n)) $\leq$ (*run-tick-count sub* c_1 n))›
by *simp*
hence
1:‹($\neg$*hamlet* ((*Rep-run sub*) *0* c_2)) $\wedge$ ($\forall n{::}nat.$ (*tick-count sub* c_2 (*Suc* n)) $\leq$ (*tick-count sub* c_1 n))›
by (*simp add*: *tick-count-is-fun*)
have ‹$\forall n{::}nat.$ (*tick-count* r c_2 (*Suc* n)) $\leq$ (*tick-count* r c_1 n)›
proof –
{ fix $n{::}nat$
have ‹*tick-count* r c_2 (*Suc* n) $\leq$ *tick-count* r c_1 n›
proof (*cases* ‹$\exists n_0.$ f $n_0 = n$›)
case *True* — n is in the image of f
from *this* **obtain** n_0 **where** *fn*:‹f $n_0 = n$› **by** *blast*
show *?thesis*
proof (*cases* ‹$\exists sn_0.$ 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** ‹$tick\text{-}count\ sub\ c_2\ sn_0 \leq tick\text{-}count\ sub\ c_1\ n_0$› **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 ‹$\neg hamlet\ ((Rep\text{-}run\ r)\ (Suc\ n)\ c_2)$›
using ∗ **by** (*simp add*: *dilating-def dilating-fun-def*)
hence ‹$tick\text{-}count\ r\ c_2\ (Suc\ n) = tick\text{-}count\ r\ c_2\ n$› **by** (*simp add*: *tick-count-suc*)
also have ‹$\ldots = tick\text{-}count\ sub\ c_2\ n_0$› **using** *fn tick-count-sub*[*OF* ∗] **by** *simp*
finally have ‹$tick\text{-}count\ r\ c_2\ (Suc\ n) = tick\text{-}count\ sub\ c_2\ n_0$› .
moreover have ‹$tick\text{-}count\ sub\ c_2\ n_0 \leq tick\text{-}count\ sub\ c_2\ (Suc\ n_0)$›
by (*simp add*: *tick-count-suc*)
ultimately have ‹$tick\text{-}count\ r\ c_2\ (Suc\ n) \leq tick\text{-}count\ sub\ c_2\ (Suc\ n_0)$› **by** *simp*
moreover have ‹$tick\text{-}count\ sub\ c_2\ (Suc\ n_0) \leq tick\text{-}count\ sub\ c_1\ n_0$› **using** *1* **by** *simp*
ultimately have ‹$tick\text{-}count\ r\ c_2\ (Suc\ n) \leq tick\text{-}count\ sub\ c_1\ n_0$› **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** ‹$tick\text{-}count\ r\ c_1\ n = tick\text{-}count\ r\ c_1\ (f\ n_p)$› .
hence *a*:‹$tick\text{-}count\ r\ c_1\ n = tick\text{-}count\ sub\ c_1\ n_p$› **using** *tick-count-sub*[*OF* ∗] **by** *simp*
have *b*: ‹$tick\text{-}count\ sub\ c_2\ (Suc\ n_p) \leq tick\text{-}count\ sub\ c_1\ n_p$› **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** ‹$tick\text{-}count\ sub\ c_2\ sn_0 \leq tick\text{-}count\ sub\ c_1\ n_p$› **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 ‹$\neg hamlet\ ((Rep\text{-}run\ r)\ (Suc\ n)\ c_2)$›
using ∗ **by** (*simp add*: *dilating-def dilating-fun-def*)
hence ‹$tick\text{-}count\ r\ c_2\ (Suc\ n) = tick\text{-}count\ r\ c_2\ n$› **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* δτ *on ms implies b* ⟧$_{TESL}$›
shows ‹*r* ∈ ⟦ *a time*−*delayed by* δτ *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*) + δτ)
⟶ *hamlet* ((*Rep-run sub*) *m b*))›
using *TESL-interpretation-atomic.simps*(*5*)[*of* ‹*a*› ‹δτ› ‹*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*) + δτ)
⟶ *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*) + δτ)
⟶ *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*) + δτ)

```
              ⟶ hamlet ((Rep-run r) (f m₀) b))› by blast
    have ‹(∀ m ≥ n. first-time r ms m (time ((Rep-run r) n ms) + δτ)
              ⟶ hamlet ((Rep-run r) m b)) ›
    proof −
    { fix m assume hyp:‹m ≥ n›
     have ‹first-time r ms m (time (Rep-run r n ms) + δτ) ⟶ hamlet (Rep-run
r m b)›
      proof (cases ‹∃ m₀. f m₀ = m›)
        case True
        from this obtain m₀ where ‹m = f m₀› 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 ‹∃ pm. m = Suc pm› by (simp add: not0-implies-Suc)
          from this obtain pm where mpm:‹m = Suc pm› by blast
          hence ‹∄ pm₀. f pm₀ = Suc pm› using ‹∄ m₀. f m₀ = 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 ‹∃ 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) + δτ))›
            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 ≪ r›
    and ‹r ∈ ⟦ time−relation ⌊c₁, c₂⌋ ∈ R ⟧TESL›
   shows ‹sub ∈ ⟦ time−relation ⌊c₁, c₂⌋ ∈ 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* ≪ *r*›
and ‹$sub \in [\![\ time\text{−}relation\ \lfloor c_1, c_2 \rfloor \in R\ ]\!]_{TESL}$›
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_0*prop*:‹$f\ n_0 = Suc\ n$› **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 ‹$\forall n.$ *hamlet* (*Rep-run r n* c_1) $\longrightarrow$ ($\forall m \geq n.$ $\neg$ *hamlet* (*Rep-run r m* c_2))›
using *ticks-imp-ticks-subk*[*OF* $*$] **by** *blast*
thus *?thesis* **using** *TESL-interpretation-atomic.simps*(*8*) **by** *blast*
qed

lemma *atomic-sub*:
assumes ‹*sub* $\ll$ *r*›
and ‹*sub* $\in$ $[\![\varphi]\!]_{TESL}$›
shows ‹*r* $\in$ $[\![\varphi]\!]_{TESL}$›
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* $\ll$ *r*›
shows ‹*sub* $\in$ $[\![[\![S]\!]\!]_{TESL} \Longrightarrow r \in [\![[\![S]\!]\!]_{TESL}$›
proof (*induction S*)
case *Nil*
thus *?case* **by** *simp*
next
case (*Cons a s*)
from *Cons.prems* **have** *sa*:‹*sub* $\in$ $[\![a]\!]_{TESL}$› **and** *sb*:‹*sub* $\in$ $[\![[\![s]\!]\!]_{TESL}$›
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.