On the Semantics of Polychronous Polytimed Specifications

Hai Nguyen Van[1][0000−0002−0585−1651], Thibaut Balabonski[1], Frédéric Boulanger[1,2][0000−0003−3185−2807], Chantal Keller[1][0000−0002−1282−0677], Benoît Valiron[1,2], and Burkhart Wolff[1]

[1] Université Paris-Saclay, CNRS, LRI, 91405, Orsay, France
[2] CentraleSupélec, France

Abstract. In this paper, we study the semantics of a specification language for the coordination of concurrent systems, which supports time at different levels: various time domains, polychrony, and mixed metric/-logical time constraints. The language itself is defined by a denotational semantics. In order to be able to construct the possible timelines for verification purposes, we also define a symbolic operational semantics, which is the reference for an efficient implementation of a tool for runtime-testing of heterogeneous systems. This study presents a novel way to link these two semantics by taking advantage of a coinductive unfolding principle of these timelines. Furthermore, these semantics and their equivalence have been formalized in the Isabelle/HOL proof assistant, together with proofs for soundness, completeness and progress.

Keywords: concurrency · coordination · semantics · timed behaviors

1 Introduction

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 differential equations, state machines, synchronous data-flow networks, discrete event models and so on. 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.

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

- events may occur in different subsystems at unrelated times, leading to *polychronous* systems [6], not necessarily under a common base clock,
- the behavior of the subsystems is observed only at a series of discrete instants,
- the instants at which a system is observed may be arbitrary and should not change its behavior (stutter-invariance),
- the coordination between subsystems involves *causality*, so the occurrence of an event may cause the occurrence of other events, possibly after a delay,

– the domain of time (discrete, rational, continuous,...) may be different in the subsystems, leading to *polytimed* systems,
– the time frames of different subsystems may be related (for instance, time in a GPS satellite and in a GPS receiver on Earth are different but related).

Figure 1 presents a heterogeneous model with subsystems modeled with a timed finite state machine, discrete events, and synchronous dataflows. To model the full system, some architectural glue is needed to coordinate these subsystems.

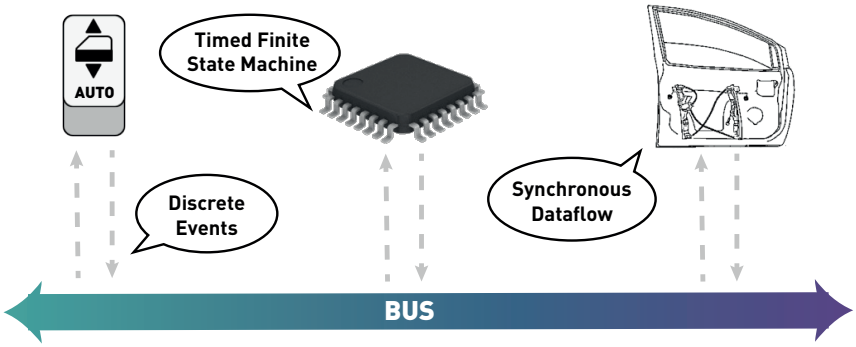


Fig. 1: The Power Window: a Heterogeneous Timed System Model

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. Also, it is always possible to add arbitrary observation instants, which allows for stuttering and modular composition of systems. Finally, a *run* is defined by a sequence of these observation instants. We can now consider the concept of *timed specification language*, which is a set of formulae that constrains the space of possible runs. This correspondence from specifications to run space is precisely a *denotational semantics*, and specifications are composed by intersecting the denoted run sets of constraint formulae.

For monitoring and online testing of heterogeneous systems, an *operational semantics* was defined in [23] to calculate concrete prefixes of runs. However, the rules of this semantics are somewhat arbitrary and not suitable for reasoning about complete runs. Our study presents a minimal specification language named TESL$^-$ (a side-effect-free subset of TESL [4]) for which our main results are:

– a denotational and an operational semantics for TESL$^-$,
– a formal validation of the operational semantics w.r.t. the denotational semantics by means of proofs for soundness, completeness and progress.

This constitutes also the outline of our paper. Compared to [23], which relied on an ad hoc operational semantics of TESL[3] implemented in Standard ML, the present work relies on properly defined and mechanized semantics. Moreover, the logical structure used for linking both semantics allows for easily-defined extensions of the language.

All definitions and theorems have been formalized into the Isabelle/HOL proof assistant [25,24] and have been accepted in the Archive of Formal Proofs, giving us a high level of confidence in our results. The latest version of the mechanized theory is available online at `github.com/heron-solver/TESL-Theory`. However, this paper is self-contained: all the key intermediate lemmas are stated using mathematical notations and their proofs are sketched.

2 TESL$^-$

We present here the TESL$^-$ minimal specification language in two parts: a basic causal one and a temporal one.

2.1 The Causality Part

Here is a grammar for the purely causal part of the language:

$$\begin{array}{lcl} \Psi & ::= & \langle atom \rangle \wedge \ \ldots \ \wedge \langle atom \rangle \\ \langle atom \rangle & ::= & \langle clock \rangle \ \texttt{sporadic} \ \langle timestamp \rangle \ \texttt{on} \ \langle clock \rangle \\ & | & \langle clock \rangle \ \texttt{implies} \ \langle clock \rangle \end{array}$$

where $\langle clock \rangle \in \mathbb{K}$ (set of clocks), and $\langle timestamp \rangle \in \mathbb{T}$ (domain of timestamps). The meaning of a specification Ψ and of the atomic formulae are as follows:

- the composition of specifications is their conjunction $\wedge$,
- a `sporadic on` requires a tick on the first clock, at an instant where the time has the specified value in the time frame of the second clock[4],
- an `implies` atom models instantaneous causality. It specifies that in every instant, if the first clock ticks, the second ticks too.

In order to define the semantics of the above syntax, we formally define the idea of runs and instants as previously introduced. The set of *runs* is defined by a clock-indexed Kripke model: $\mathsf{Runs} = \mathbb{N} \to \mathbb{K} \to (\mathbb{B} \times \mathbb{T})$, where $\mathbb{K}$ 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 $\mathbb{T}$ is a universal metric time space with some linear ordering $\leq_{\mathbb{T}}$. Also, we constrain this run space to prevent time from flowing backwards, in other words a run $\rho \in \mathsf{Runs}$ must be *monotonic*[5]:

$$\forall n \in \mathbb{N}.\ \forall C \in \mathbb{K}.\quad \pi_2(\rho\ n\ C) \ \leq_{\mathbb{T}} \ \pi_2(\rho\ (n+1)\ C)$$

[3] `wdi.centralesupelec.fr/software/TESL/`

[4] The two clocks in `sporadic on` may be identical, which means that this clock must tick at the given time stamp.

[5] π_2 being the second projection, with $\pi_2(x, y) = y$

A run is simply a infinite-sequence of *instants*. From a position n and a run ρ, we can extract the instant $\rho_n \in \mathbb{K} \to (\mathbb{B} \times \mathbb{T})$. Instants describe the status of each clock at a given observation. We define two projections to get the status of a clock C at an instant ρ_n:

- $\mathsf{ticks}(\rho_n(C))$ indicates whether C ticks
- $\mathsf{time}(\rho_n(C))$ is the timestamp of C at that instant.

The denotation $[\![\Psi]\!]_{\mathsf{TESL}}$ of a TESL$^-$ formula Ψ is defined inductively as follows:

$$[\![\psi_0 \wedge \ldots \wedge \psi_k]\!]_{\mathsf{TESL}} \stackrel{\text{def}}{=} [\![\psi_0]\!]_{\mathsf{TESL}} \cap \ldots \cap [\![\psi_k]\!]_{\mathsf{TESL}}$$

$$[\![C_1 \texttt{ sporadic } \tau \texttt{ on } C_2]\!]_{\mathsf{TESL}} \stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \exists n \in \mathbb{N}.\ \mathsf{ticks}(\rho_n(C_1)) \wedge \mathsf{time}(\rho_n(C_2)) = \tau\}$$

$$[\![C_{\mathsf{master}} \texttt{ implies } C_{\mathsf{slave}}]\!]_{\mathsf{TESL}} \stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \forall n \in \mathbb{N}.\ \mathsf{ticks}(\rho_n(C_{\mathsf{master}})) \Longrightarrow \mathsf{ticks}(\rho_n(C_{\mathsf{slave}}))\}$$

2.2 The Temporal Part

We introduce here some operators concerning time and duration.

$$\begin{array}{rcl} \langle atom \rangle & ::= & \ldots \\ & \mid & \texttt{time relation (}\langle clock \rangle\texttt{, }\langle clock \rangle\texttt{) } \in \langle relation \rangle \\ & \mid & \langle clock \rangle \texttt{ time delayed by } \langle duration \rangle \texttt{ on } \langle clock \rangle \texttt{ implies } \langle clock \rangle \end{array}$$

where $\langle relation \rangle \subseteq \mathbb{T} \times \mathbb{T}$ and $\langle duration \rangle \in \mathbb{T}$. The meaning of these operators is:

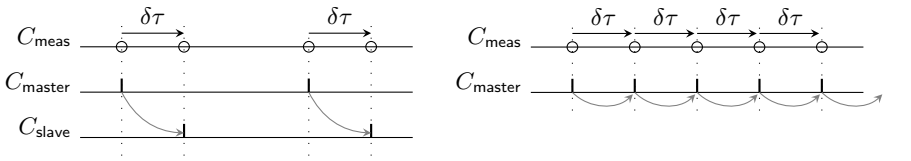
- a `time relation` atom gives a relation between the time frames of two clocks. The time stamps of the clocks must be in the relation at every instant,
- a `time delayed` atom represents delayed causality. When the first (master) clock ticks, the duration is added to the current time on the second (measuring) clock to obtain the date at which the third (slave) clock has to tick.

The denotation of these operators is as follows:

$$[\![\texttt{time relation (}C_1\texttt{, }C_2\texttt{) } \in R]\!]_{\mathsf{TESL}} \stackrel{\text{def}}{=} \left\{\rho \in \mathsf{Runs} \;\middle|\; \forall n \in \mathbb{N}.\ \left(\mathsf{time}(\rho_n(C_1)), \mathsf{time}(\rho_n(C_2))\right) \in R\right\}$$

$$[\![C_{\mathsf{master}} \texttt{ time delayed by } \delta\tau \texttt{ on } C_{\mathsf{meas}} \texttt{ implies } C_{\mathsf{slave}}]\!]_{\mathsf{TESL}} \stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \forall n \in \mathbb{N}.\ \mathsf{ticks}(\rho_n(C_{\mathsf{master}})) \Longrightarrow \forall m \geq n.\ \mathsf{time}(\rho_m(C_{\mathsf{meas}})) = \mathsf{time}(\rho_n(C_{\mathsf{meas}})) + \delta\tau \Longrightarrow \mathsf{ticks}(\rho_m(C_{\mathsf{slave}}))\}$$

A time relation makes it possible to specify, for example, that time on one clock flows 2.5 times as fast as on another clock. Using the floor function $\lfloor \_ \rfloor$, it is also possible to establish a relation between continuous and discrete time frames.



(a) C_{master} `time delayed by` $\delta\tau$ `on` C_{meas} `implies` C_{slave} (b) C_{master} `time delayed by` $\delta\tau$ `on` C_{meas} `implies` C_{master}

Fig. 2: Time delays and periodicity

The `time delayed` construct introduces *durations* in causal relationships. Figure 2a shows the causal relation between C_{master} and C_{slave}, and the duration measured in the time frame of clock C_{meas}. Figure 2b shows how to specify a periodic clock using this construct. Notice that there are no ticks on clock C_{meas} in this example, it is only used as a time frame to measure durations.

2.3 An Application Example: the Car Power Window

The car power window [3], illustrated in Figure 1, is an example of timed coordination of four subsystems: a control button, a timed finite state machine, a synchronous data flow (SDF) model of the electro-mechanical parts, and a discrete events (DE) model of the CAN bus, which interconnects the other subsystems.

For the sake of brevity, we consider only the raising of the window. Therefore, the button can only be pulled up and released, what we model by the `btn_up` and `btn_neutral` events. Similarly, we consider the `up` and `stop` input events for the timed finite state machine, as well as its `power` output event, which denotes the sending of a power command to the electromechanical subsystem (the value of this command is ignored in our temporal coordination model).

The model of the electromechanical subsystem has an `update_power` input, which corresponds to an update of the power to deliver to the motor. However, according to the SDF nature of this subsystem, this information is only taken into account when it reacts to compute its next state, which occurs every 50 ms and is modeled by a `react` input event. This periodic activation is part of the design of this subsystem, and it must be enforced for the regulation of the current in the motor to work properly. Here is a TESL specification for the power window:

```
1  unit-clock btn_up       // the button is pulled up
2  unit-clock btn_neutral  // the button is released
3  unit-clock up           // the TFSM receives an up event
4  unit-clock stop         // the TFSM receives a stop event
5  unit-clock power        // the TFSM produces a power event
6  unit-clock update_power // the SDF model gets a new power command
7  unit-clock react        // the SDF model reacts to its inputs
8  rational-clock realtime // real-time in seconds
9  rational-clock bus      // time scale of the CAN bus
10
11 time relation realtime = 0.002 * bus
12 btn_up time delayed by 1.0 on bus implies up
```

```
13 btn_neutral time delayed by 1.0 on bus implies stop
14
15 // Inputs of the TFSM trigger an instantaneous update of its output
16 up implies power
17 stop implies power
18
19 // The transmission delay on the CAN bus is 2 ms
20 power time delayed by 1.0 on bus implies update_power
21
22 // The window must react every 50ms (periodic clock)
23 react time delayed by 0.05 on realtime implies react
```

This specification ignores the values that are sent over the bus, it specifies only when things happen since its goal is to coordinate the behaviors of the subsystems. Lines 1 to 7 declare the clocks that compose the interface of the subsystems for the architectural glue, as explained on Figure 1. The `unit-clock` keyword simply sets the domain of timestamps of these clocks to a single value. Lines 8 and 9 declare chronometric clocks used to measure elapsed time on the CAN bus and in the real world. Their time domain is the rationals. Line 11 is an example of a relation between two time frames. It specifies that when 1 unit of time elapses on the bus clock, 0.002 s elapses on the real time clock, which means that time is measured in units of 2 ms on the bus clock. Lines 12 and 13 specify that when the button is pulled up or released, the timed finite state machine receives its `up` or `stop` input event 1 unit of time later, measured in the time frame of the bus clock (2 ms in real time). Lines 16 and 17 specify that the state machine reacts instantaneously to its inputs by producing its `power` output event. Line 20 specifies the transmission delay on the bus between the state machine and the SDF subsystem. Last, line 23 specifies that the reaction of the SDF subsystem is periodic, because it implies itself with a delay of 50 ms.

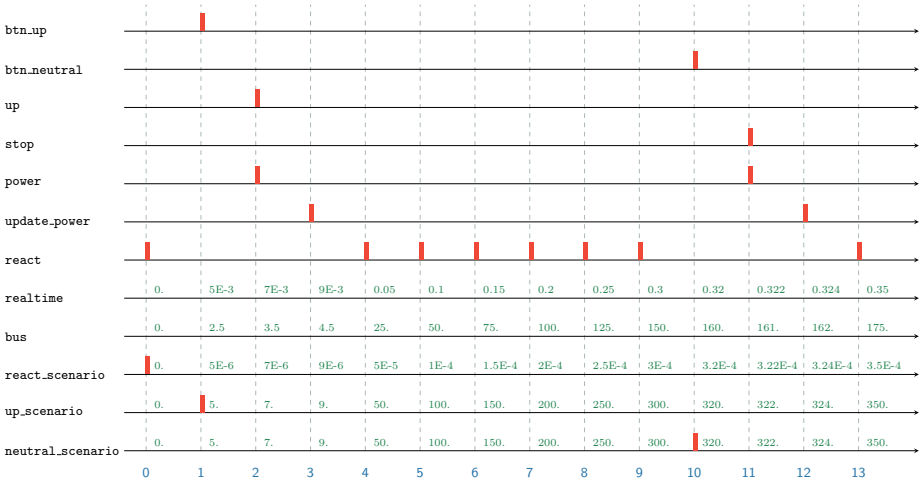


Fig. 3: A satisfying run for the example of the power window specification

Figure 3 depicts a satisfying run. The user pulls the button up (clock `btn_up`) at 5ms (on the time scale of the `realtime` clock). The controller receives this information (clock `up`) at 7ms due to the transmission delay on the CAN bus, and immediately sets the power for the window motor (clock `power`). Then, the mechanical part receives the command at 9ms (clock `update_power`). The next tick of the periodic `react` clock occurs at 50ms, which is the time at which the new value of the power is taken into account and the window starts moving up. At 320ms, the user releases the button, which switches back to neutral (clock `btn_neutral`). The new value of the power is updated at 324ms because of the transmission delays between the button and the controller, and between the controller and the mechanical parts. The next reaction of the window (clock `react`) occurs at 350ms, which is the time at which the window stops moving up. The additional clocks `react_scenario`, `up_scenario` and `neutral_scenario` are used to describe the user interface simulation scenario.

2.4 Properties of the Semantics

An important property that we derive directly from the denotational semantics is invariance by stuttering. When we combine two specifications Ψ_1 and Ψ_2, clocks in Ψ_2 may tick at instants where no clock in Ψ_1 ticks. Therefore, runs that satisfy S_1 should still satisfy it when these stuttering instants are added. Other specification languages, such as LTL [17,16], seek stutter-invariance to avoid the exponential explosion of the search space when checking properties [12,15,7,21]. In TESL, this idea is fully explored in the mechanized theory as previously mentioned.

3 Operational Semantics

We define an operational semantics to be able to constructively derive all possible satisfying runs for a given specification. This operational semantics works on configurations, which are composed of three parts informally called the *past*, the *present* and the *future*. The semantic rules unfold the constraints of the future into the present, and the non-deterministic choices that are made in the present are then stored into the past. The decisions on the past are expressed using *primitive constraints* defined in subsection 3.1. The combination of the past, the present and the future is called a *configuration*, as presented in subsection 3.2. The reduction rules on configurations are presented in subsection 3.3.

3.1 Primitives

The *primitives* in Definition 2 describe prefixes of satisfying runs. Note that compared to TESL$^-$ atomic formulae, they deal with fixed instant indexes.

Definition 1 (Time Variables). *The set of time variables* $\mathbb{V}$ *contains symbols* tvar_n^C *with* $n \in \mathbb{N}$ *and* $C \in \mathbb{K}$. *Note that* tvar_n^C *stands for the symbolic value of time on clock* C *at instant* n.

Definition 2 (Run Primitives). A run primitive $\gamma \in \Gamma$ is one of:

- $C \Uparrow_n$ constrains clock C to tick at instant index n ;
- $C \not\Uparrow_n$ constrains clock C not to tick (to be idle) at instant index n;
- $C \Downarrow_n x$ constrains clock C to have timestamp x at instant index n, where x can be a variable in $\mathbb{V}$, or a constant in $\mathbb{T}$;
- $(\mathsf{tvar}_{n_1}^{C_1},\ \mathsf{tvar}_{n_2}^{C_2}) \in R$ constrains values $\mathsf{tvar}_{n_1}^{C_1}$ and $\mathsf{tvar}_{n_2}^{C_2}$ to be in relation R.

The semantics of these primitives is given by $[\![\_]\!]_{\mathsf{prim}}$ as:

$$[\![\{\gamma_0\ ;\ \dots\ ;\ \gamma_k\}]\!]_{\mathsf{prim}} \stackrel{\text{def}}{=} [\![\gamma_0]\!]_{\mathsf{prim}} \cap \dots \cap [\![\gamma_k]\!]_{\mathsf{prim}}$$

$$[\![C \Uparrow_n]\!]_{\mathsf{prim}} \stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \mathsf{ticks}(\rho_n(C)) \text{ is true}\}$$

$$[\![C \not\Uparrow_n]\!]_{\mathsf{prim}} \stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \mathsf{ticks}(\rho_n(C)) \text{ is false}\}$$

$$[\![C \Downarrow_n x]\!]_{\mathsf{prim}} \stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \mathsf{time}(\rho_n(C)) = x\} \text{ with } x \text{ in } \mathbb{T} \text{ or } \mathbb{V}$$

$$[\![(\mathsf{tvar}_{n_1}^{C_1},\ \mathsf{tvar}_{n_2}^{C_2}) \in R]\!]_{\mathsf{prim}} \stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \mathsf{time}(\rho_{n_1}(C_1)) \text{ and } \mathsf{time}(\rho_{n_2}(C_2)) \text{ are in } R\}$$

3.2 Configurations

The operational semantics transforms *configurations*, which represent the "current" state of the construction of a symbolic run and have three parts:

- the past Γ is a collection of primitive constraints that represents what has been decided in previous instants (the prefix of the run);
- the present Ψ contains the constraints on the instant under scrutiny (what can or cannot be added to the prefix);
- the future Φ contains the constraints on the future behavior of the run.

Definition 3 (Configuration). A *configuration* is a tuple $\Gamma \models_n \Psi \triangleright \Phi$, where n is the index of the current instant, Γ the context, which contains primitives describing the "past", Ψ the TESL$^-$-formula to be considered in the "present", and Φ the TESL$^-$-formula to satisfy in the "future" of the run.

3.3 Reduction Rules

The semantics consists in rules that transform configurations in two ways:

1. Moving constraints from the future to the present (introduction), which amounts to turning the "next" instant into the current instant;
2. Consuming constraints in the present to produce primitive constraints in the past (elimination).

The introduction rule initializes a new instant by incrementing the index counter and moving the constraints from the future into the present.

Definition 4 (Introduction Rule $\rightarrow_i$). The relation $\rightarrow_i$ is the smallest relation satisfying:

$$\Gamma \models_n \varnothing \triangleright \Phi \quad \rightarrow_i \quad \Gamma \models_{n+1} \Phi \triangleright \varnothing \qquad (\mathsf{instant}_i)$$

$$\Gamma \models_n \Psi \wedge (C_1 \texttt{ sporadic } \tau \texttt{ on } C_2) \triangleright \Phi \qquad (\mathsf{sporadic-on}_{e1})$$
$$\rightarrow_e \Gamma \models_n \Psi \triangleright \Phi \wedge (C_1 \texttt{ sporadic } \tau \texttt{ on } C_2)$$
$$\Gamma \models_n \Psi \wedge (C_1 \texttt{ sporadic } \tau \texttt{ on } C_2) \triangleright \Phi \qquad (\mathsf{sporadic-on}_{e2})$$
$$\rightarrow_e \Gamma \cup \{C_1 \Uparrow_n,\ C_2 \Downarrow_n \tau\} \models_n \Psi \triangleright \Phi$$
$$\Gamma \models_n \Psi \wedge (C_{\mathsf{master}} \texttt{ implies } C_{\mathsf{slave}}) \triangleright \Phi \qquad (\mathsf{implies}_{e1})$$
$$\rightarrow_e \Gamma \cup \{C_{\mathsf{master}} \not\Uparrow_n\} \models_n \Psi \triangleright \Phi \wedge (C_{\mathsf{master}} \texttt{ implies } C_{\mathsf{slave}})$$
$$\Gamma \models_n \Psi \wedge (C_{\mathsf{master}} \texttt{ implies } C_{\mathsf{slave}}) \triangleright \Phi \qquad (\mathsf{implies}_{e2})$$
$$\rightarrow_e \Gamma \cup \{C_{\mathsf{master}} \Uparrow_n,\ C_{\mathsf{slave}} \Uparrow_n\} \models_n \Psi \triangleright \Phi \wedge (C_{\mathsf{master}} \texttt{ implies } C_{\mathsf{slave}})$$
$$\Gamma \models_n \Psi \wedge (\texttt{time relation } (C_1, C_2) \in R) \triangleright \Phi \qquad (\mathsf{time-relation}_e)$$
$$\rightarrow_e \Gamma \cup \{(\mathsf{tvar}_n^{C_1},\ \mathsf{tvar}_n^{C_2}) \in R\} \models_n \Psi \triangleright \Phi \wedge (\texttt{time relation } (C_1, C_2) \in R)$$
$$\Gamma \models_n \Psi \wedge (C_{\mathsf{master}} \texttt{ time delayed by } \delta t \texttt{ on } C_{\mathsf{meas}} \texttt{ implies } C_{\mathsf{slave}}) \triangleright \Phi \qquad (\mathsf{time-delayed}_{e1})$$
$$\rightarrow_e \Gamma \cup \{C_{\mathsf{master}} \not\Uparrow_n\} \models_n \Psi \triangleright \Phi \wedge (C_{\mathsf{master}} \texttt{ time delayed by } \delta t \texttt{ on } C_{\mathsf{meas}} \texttt{ implies } C_{\mathsf{slave}})$$
$$\Gamma \models_n \Psi \wedge (C_{\mathsf{master}} \texttt{ time delayed by } \delta t \texttt{ on } C_{\mathsf{meas}} \texttt{ implies } C_{\mathsf{slave}}) \triangleright \Phi \qquad (\mathsf{time-delayed}_{e2})$$
$$\rightarrow_e \Gamma \cup \{C_{\mathsf{master}} \Uparrow_n\} \models_n \Psi \wedge (C_{\mathsf{slave}} \texttt{ sporadic } (\mathsf{tvar}_n^{C_{\mathsf{meas}}} + \delta t) \texttt{ on } C_{\mathsf{meas}})$$
$$\triangleright \quad \Phi \wedge (C_{\mathsf{master}} \texttt{ time delayed by } \delta t \texttt{ on } C_{\mathsf{meas}} \texttt{ implies } C_{\mathsf{slave}})$$

Table 1: Elimination Rules for TESL$^-$ formulae

The elimination rules consume constraints on the present and produce primitive constraints on the past as well as constraints on the future, which correspond to consequences of the choices made for the current instant. The application of these rules adds constraints to Γ and makes the run more defined.

Definition 5 (Elimination Rules $\rightarrow_e$). The relation $\rightarrow_e$ is the smallest relation satisfying the rules given in Table 1.

It is necessary to apply elimination rules until the present of the configuration is empty and the introduction rule can be applied to progress to the next instant. Here are different possibilities to eliminate constraints from the present:

- C_1 `sporadic` τ `on` C_2: this formula can be postponed to a later instant (Rule $\mathsf{sporadic-on}_{e1}$), or satisfied in the current instant by adding ticking and timestamp primitives to the context (Rule $\mathsf{sporadic-on}_{e2}$),
- C_{master} `implies` C_{slave}: either clock C_{master} does not tick (Rule $\mathsf{implies}_{e1}$), or both C_{master} and C_{slave} tick in the current instant (Rule $\mathsf{implies}_{e2}$). In both cases, the formula is copied into the future to be satisfied at every instant,
- `time relation` $(C_1, C_2) \in R$: the corresponding primitive is added to constrain the timestamps on clocks C_1 and C_2 at the current instant, and the formula is put into the future since it has to be satisfied at every instant,
- C_{master} `time delayed by` δt `on` C_{meas} `implies` C_{slave}: either clock C_{master} does not tick and we only copy the formula into the future (Rule $\mathsf{time-delayed}_{e1}$);

$$[\![\psi_0 \wedge \ldots \wedge \psi_k]\!]^{\geq i}_{\mathsf{TESL}} \stackrel{\text{def}}{=} [\![\psi_0]\!]^{\geq i}_{\mathsf{TESL}} \cap \ldots \cap [\![\psi_k]\!]^{\geq i}_{\mathsf{TESL}}$$

$$[\![C_1 \texttt{ sporadic } \tau \texttt{ on } C_2]\!]^{\geq i}_{\mathsf{TESL}}$$
$$\stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \exists n \geq i.\ \mathsf{ticks}(\rho_n(C_1)) \wedge \mathsf{time}(\rho_n(C_2)) = \tau\}$$

$$[\![C_{\mathsf{master}} \texttt{ implies } C_{\mathsf{slave}}]\!]^{\geq i}_{\mathsf{TESL}}$$
$$\stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \forall n \geq i.\ \mathsf{ticks}(\rho_n(C_{\mathsf{master}})) \implies \mathsf{ticks}(\rho_n(C_{\mathsf{slave}}))\}$$

$$[\![\texttt{time relation (}C_1\texttt{, }C_2\texttt{) } \in R]\!]^{\geq i}_{\mathsf{TESL}}$$
$$\stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \forall n \geq i.\ (\mathsf{time}(\rho_n(C_1)), \mathsf{time}(\rho_n(C_2))) \in R\}$$

$$[\![C_{\mathsf{master}} \texttt{ time delayed by } \delta\tau \texttt{ on } C_{\mathsf{meas}} \texttt{ implies } C_{\mathsf{slave}}]\!]^{\geq i}_{\mathsf{TESL}}$$
$$\stackrel{\text{def}}{=} \{\rho \in \mathsf{Runs} \mid \forall n \geq i.\ \mathsf{ticks}(\rho_n(C_{\mathsf{master}})) \implies$$
$$\forall m \geq n.\ \mathsf{time}(\rho_m(C_{\mathsf{meas}})) = \mathsf{time}(\rho_n(C_{\mathsf{meas}})) + \delta\tau \implies \mathsf{ticks}(\rho_m(C_{\mathsf{slave}}))\}$$

Table 2: Stepwise Interpretation of TESL$^-$ formulae

or it ticks and we need to force a tick on C_{slave} when the time on C_{meas} reaches $\mathsf{tvar}_n^{C_{\mathsf{meas}}} + \delta t$, which is the current timestamp on measuring clock C_{meas} delayed by duration δt. The formula is copied into the future (Rule $\mathsf{time} - \mathsf{delayed}_{e2}$).

3.4 Local Termination

Proposition 1 (Termination of Elimination Rules).
The relation $\rightarrow_e$ is well-founded.

Proof. All of the elimination rules strictly decrease the number of formulae in the "present" of the configuration, and a configuration with an empty "present" is in normal form with respect to $\rightarrow_e$.

Definition 6 (Reduction $\rightarrow$). We define $\rightarrow \stackrel{\text{def}}{=} \rightarrow_i \cup \rightarrow_e$.
A reduction step is either an introduction or an elimination.

4 Relating Operational and Denotational Semantics

In this section, we give key properties of the operational semantics. We are interested in establishing soundness (Theorem 1), completeness (Theorem 2), and progress (Theorem 3) with respect to the denotational semantics defined in section 2.

4.1 Stepwise Denotational Semantics

In subsection 2.1 and subsection 2.2, we have defined a denotational semantics to characterize the runs that satisfy a specification. Definition 7 gives a stepwise version of this definition, which constrains the behavior only from a given instant.

$$
\begin{aligned}
&[\![C_1 \texttt{ sporadic } \tau \texttt{ on } C_2]\!]^{\geq i}_{\mathsf{TESL}} \\
&\quad = \left([\![C_1 \Uparrow_i]\!]_{\mathsf{prim}} \cap [\![C_2 \Downarrow_i \tau]\!]_{\mathsf{prim}}\right) \quad \cup \quad [\![C_1 \texttt{ sporadic } \tau \texttt{ on } C_2]\!]^{\geq i+1}_{\mathsf{TESL}} \\
&[\![C_{\mathsf{master}} \texttt{ implies } C_{\mathsf{slave}}]\!]^{\geq i}_{\mathsf{TESL}} \\
&\quad = \left([\![C_{\mathsf{master}} \not\Uparrow_i]\!]_{\mathsf{prim}} \quad \cup \quad ([\![C_{\mathsf{master}} \Uparrow_i]\!]_{\mathsf{prim}} \cap [\![C_{\mathsf{slave}} \Uparrow_i]\!]_{\mathsf{prim}})\right) \\
&\quad \cap \quad [\![C_{\mathsf{master}} \texttt{ implies } C_{\mathsf{slave}}]\!]^{\geq i+1}_{\mathsf{TESL}} \\
&[\![\texttt{time relation } (C_1, C_2) \in R]\!]^{\geq i}_{\mathsf{TESL}} \\
&\quad = [\![(\mathsf{tvar}^i_{C_1}, \mathsf{tvar}^i_{C_2}) \in R]\!]_{\mathsf{prim}} \quad \cap \quad [\![\texttt{time relation } (C_1, C_2) \in R]\!]^{\geq i+1}_{\mathsf{TESL}} \\
&[\![C_{\mathsf{master}} \texttt{ time delayed by } \delta\tau \texttt{ on } C_{\mathsf{meas}} \texttt{ implies } C_{\mathsf{slave}}]\!]^{\geq i}_{\mathsf{TESL}} \\
&\quad = [\![C_{\mathsf{master}} \not\Uparrow_i]\!]_{\mathsf{prim}} \cap [\![C_{\mathsf{master}} \texttt{ time delayed by } \delta\tau \texttt{ on } C_{\mathsf{meas}} \texttt{ implies } C_{\mathsf{slave}}]\!]^{\geq i+1}_{\mathsf{TESL}} \\
&\quad \cup \quad [\![C_{\mathsf{master}} \Uparrow_i]\!]_{\mathsf{prim}} \cap [\![C_{\mathsf{slave}} \texttt{ sporadic } \mathsf{tvar}^i_{C_{\mathsf{meas}}} + \delta\tau \texttt{ on } C_{\mathsf{meas}}]\!]^{\geq i}_{\mathsf{TESL}} \\
&\qquad\qquad\qquad \cap [\![C_{\mathsf{master}} \texttt{ time delayed by } \delta\tau \texttt{ on } C_{\mathsf{meas}} \texttt{ implies } C_{\mathsf{slave}}]\!]^{\geq i+1}_{\mathsf{TESL}}
\end{aligned}
$$

Table 3: Coinductive Unfolding of Stepwise Interpretation

Definition 7 (Stepwise Interpretation of TESL$^-$ formulae).
The *stepwise interpretation* of a TESL$^-$ formula Ψ, noted $[\![\Psi]\!]^{\geq i}_{\mathsf{TESL}}$, is defined as in Table 2.

This stepwise interpretation from instant 0 matches the denotational interpretation:

Lemma 1 (Start step). *For any TESL$^-$ formula Ψ, $[\![\Psi]\!]_{\mathsf{TESL}} = [\![\Psi]\!]^{\geq 0}_{\mathsf{TESL}}$.*

Proof. From the definitions of $[\![\Psi]\!]^{\geq 0}_{\mathsf{TESL}}$ and $[\![\Psi]\!]_{\mathsf{TESL}}$ and from $n \in \mathbb{N} \iff n \geq 0$.

The next proposition links the operational and denotational semantics. The structure of the right hand term in the equations in Table 3 matches the reduction rules of the operational semantics. Therefore, the coinductive unfolding of the denotational semantics is similar to the derivation of a reduction step in the operational semantics. The past-present-future pattern is also visible here: the past is described by $[\![\text{-}]\!]_{\mathsf{prim}}$ (denotation of fixed primitives), the present by $[\![\text{-}]\!]^{\geq i}_{\mathsf{TESL}}$, which denotes runs that are valid from the current instant, and the future by $[\![\text{-}]\!]^{\geq i+1}_{\mathsf{TESL}}$, which denotes runs that are valid from the next instant.

Proposition 2 (Coinductive Unfolding). *The stepwise interpretation can be coinductively unfolded as presented in Table 3.*

Proof. By unfolding the quantifiers and substituting parts with Definition 2 and Definition 7. The rules of Table 3 state that $[\![\Psi]\!]^{\geq i}_{\mathsf{TESL}}$ can be decomposed into what happens at index i and what happens starting from index $i+1$.

This *coinductive* pattern explains the behavior of the operational semantics at a denotational level and bridges the gap between those semantics.

4.2 Soundness

To establish *soundness*, we define the meaning of a configuration.

Definition 8 (Interpretation of configurations).
The interpretation of a configuration $\Gamma \models_n \Psi \triangleright \Phi$ *is:*

$$\left[\!\left[\Gamma \models_n \Psi \triangleright \Phi\right]\!\right]_{\mathsf{config}} \stackrel{\text{def}}{=} \left[\!\left[\Gamma\right]\!\right]_{\mathsf{prim}} \cap \left[\!\left[\Psi\right]\!\right]_{\mathsf{TESL}}^{\geq n} \cap \left[\!\left[\Phi\right]\!\right]_{\mathsf{TESL}}^{\geq n+1}$$

It is trivial to show that the interpretation of a TESL$^-$ formula Ψ is the same as the interpretation of the initial configuration starting at Ψ.

Lemma 2 (Start configuration). *For any TESL$^-$ formula* Ψ*, we have*

$$\left[\!\left[\Psi\right]\!\right]_{\mathsf{TESL}} = \left[\!\left[\varnothing \models_0 \Psi \triangleright \varnothing\right]\!\right]_{\mathsf{config}}$$

Proof. The proof is done by unfolding Definition 8: $\left[\!\left[\varnothing\right]\!\right]_{\mathsf{prim}}$ and $\left[\!\left[\varnothing\right]\!\right]_{\mathsf{TESL}}^{\geq n+1}$ are the whole set of runs, since $\varnothing$ is not constraining anything, and $\left[\!\left[\Psi\right]\!\right]_{\mathsf{TESL}}^{\geq 0}$ is $\left[\!\left[\Psi\right]\!\right]_{\mathsf{TESL}}$ by Lemma 1.

We now show that each reduction step is sound, in the sense that if a run satisfies a derived configuration, it also satisfies the original configuration.

Lemma 3 (Sound Reduction). *For any reduction* $(\Gamma \models_n \Psi \triangleright \Phi) \rightarrow (\Gamma' \models_{n'} \Psi' \triangleright \Phi')$*, we have* $\left[\!\left[\Gamma \models_n \Psi \triangleright \Phi\right]\!\right]_{\mathsf{config}} \supseteq \left[\!\left[\Gamma' \models_{n'} \Psi' \triangleright \Phi'\right]\!\right]_{\mathsf{config}}$.

Proof. By Definitions 6 and 8, and case analysis on $\rightarrow$. In the $\rightarrow_i$ case, the reduction is of the form $\Gamma \models_n \Psi \triangleright \varnothing \rightarrow \Gamma \models_{n+1} \varnothing \triangleright \Psi$: the semantics of both sides are the same. In $\rightarrow_e$ case, $n' = n+1$ and we use Proposition 2 to decompose the semantics at instant n using the semantics at instant $n+1$.

Finally, we show soundness by generalizing Lemma 2 and Lemma 3 to an arbitrary number of reductions from the initial configuration.

Theorem 1 (Soundness). *Let* Ψ *be a TESL$^-$ formula. For all* k *and all configurations* $\Gamma' \models_{n'} \Psi' \triangleright \Phi'$ *such that* $\varnothing \models_0 \Psi \triangleright \varnothing \rightarrow^k \Gamma' \models_{n'} \Psi' \triangleright \Phi'$*, we have*

$$\left[\!\left[\Psi\right]\!\right]_{\mathsf{TESL}} \supseteq \left[\!\left[\Gamma' \models_{n'} \Psi' \triangleright \Phi'\right]\!\right]_{\mathsf{config}}$$

Proof. By induction on k. For the base case, when $k = 0$ we have $\Gamma' = \Psi' = \varnothing$. Lemma 2 then tells us that $\left[\!\left[\Psi\right]\!\right]_{\mathsf{TESL}} = \left[\!\left[\Gamma' \models_{n'} \Psi' \triangleright \Phi'\right]\!\right]_{\mathsf{config}}$. For the inductive case, we suppose that the result is true for k and we consider $k+1$ reductions:

$$\varnothing \models_0 \Psi \triangleright \varnothing \rightarrow^k \Gamma' \models_{n'} \Psi' \triangleright \Phi' \rightarrow \Gamma'' \models_{n''} \Psi'' \triangleright \Phi''.$$

The induction hypothesis tells us that $\left[\!\left[\Psi\right]\!\right]_{\mathsf{TESL}} \supseteq \left[\!\left[\Gamma' \models_{n'} \Psi' \triangleright \Phi'\right]\!\right]_{\mathsf{config}}$, and we can conclude using Lemma 3 and the transitivity of $\supseteq$.

4.3 Completeness

Completeness consists in showing that if a run ρ belongs to the denotation of a configuration, it is always possible to derive a new configuration whose denotation also contains ρ. For this, we first define the direct successors of a configuration.

Definition 9 (Direct Successors). For any configuration $\Gamma \models_n \Psi \triangleright \Phi$,
$\mathcal{C}_{\text{next}}(\Gamma \models_n \Psi \triangleright \Phi) \stackrel{\text{def}}{=} \left\{ \Gamma' \models_{n'} \Psi' \triangleright \Phi' \;\middle|\; (\Gamma \models_n \Psi \triangleright \Phi) \rightarrow (\Gamma' \models_{n'} \Psi' \triangleright \Phi') \right\}$

Then we show that any denoted run belongs to some successor configuration.

Lemma 4 (Complete Direct Successors). *For any configuration $\Gamma \models_n \Psi \triangleright \Phi$,*

$$\left[\!\left[\Gamma \models_n \Psi \triangleright \Phi\right]\!\right]_{\mathsf{config}} \subseteq \bigcup_{X \in \mathcal{C}_{\text{next}}(\Gamma \models_n \Psi \triangleright \Phi)} \left[\!\left[X\right]\!\right]_{\mathsf{config}}$$

Proof. Similarly to the proof of Lemma 3, we proceed by induction on the number of formulae in Ψ. If Ψ is empty, the only possible reduction is $\rightarrow_i$: the reduction is of the form $\Gamma \models_n \Psi \triangleright \varnothing \rightarrow \Gamma \models_{n+1} \varnothing \triangleright \Psi$ and there is only one possible X, whose semantics is $\left[\!\left[\Gamma \models_n \Psi \triangleright \Phi\right]\!\right]_{\mathsf{config}}$. If Ψ is not empty, the reduction is a $\rightarrow_e$-reduction. The case is solved using Proposition 2 to decompose the semantics at instant n using the semantics of the possible reductions at instant $n+1$.

Hence, completeness holds for an arbitrary number of reductions starting from the initial configuration.

Theorem 2 (Completeness). *Let Ψ be a TESL$^-$ formula and ρ a satisfying run,* i.e. $\rho \in \left[\!\left[\Psi\right]\!\right]_{\mathsf{TESL}}$. *For all k, there is a configuration $\Gamma' \models_{n'} \Psi' \triangleright \Phi'$ such that $\varnothing \models_0 \Psi \triangleright \varnothing \rightarrow^k \Gamma' \models_{n'} \Psi' \triangleright \Phi'$ and $\rho \in \left[\!\left[\Gamma' \models_{n'} \Psi' \triangleright \Phi'\right]\!\right]_{\mathsf{config}}$*

Proof. By induction on k. For $k = 0$, we conclude using Lemma 2. For the inductive case, we assume that the result is true for k and consider the $k+1$ case. From the induction hypothesis we find a configuration $\Gamma' \models_{n'} \Psi' \triangleright \Phi'$ such that $\varnothing \models_0 \Psi \triangleright \varnothing \rightarrow^k \Gamma' \models_{n'} \Psi' \triangleright \Phi'$ and $\rho \in \left[\!\left[\Gamma' \models_{n'} \Psi' \triangleright \Phi'\right]\!\right]_{\mathsf{config}}$. From Lemma 4, we deduce that there is some $X \in \mathcal{C}_{\text{next}}(\Gamma \models_n \Psi \triangleright \Phi)$ such that $\rho \in \left[\!\left[X\right]\!\right]_{\mathsf{config}}$. This X is the configuration we are looking for to close the inductive case.

4.4 Progress

Progress ensures the increase of the length of the run in construction. We establish that for any instant index, a configuration can be "executed" to produce a run prefix whose length is incremented by 1 (Lemma 5). Then in Theorem 3 we show that for any instant index, a specification can be "executed" to produce a run prefix of such length from the initial configuration.

Lemma 5 (Instant Index Increase). *Let $\Gamma \models_n \Psi \triangleright \Phi$ be a configuration and ρ a satisfying run,* i.e. $\rho \in \left[\!\left[\Gamma \models_n \Psi \triangleright \Phi\right]\!\right]_{\mathsf{config}}$. *There is Γ', Ψ', Φ' and a number of reductions k such that $\Gamma \models_n \Psi \triangleright \Phi \rightarrow^k \Gamma' \models_{n+1} \Psi' \triangleright \Phi'$ and $\rho \in \left[\!\left[\Gamma' \models_{n+1} \Psi' \triangleright \Phi'\right]\!\right]_{\mathsf{config}}$.*

Proof. By induction on the size of Ψ. When Ψ is empty, we can just pick $k = 1$ as the reduction will be a $\rightarrow_i$-reduction, and both sides of the reduction will have the same semantics. Now, supposing that the result is true for any Ψ containing i formulae, let's assume that Ψ contains $i + 1$ formulae. Lemma 4 tells us that there exists a configuration X such that $\Gamma \models_n \Psi \triangleright \Phi \rightarrow X$ and $\rho \in [\![X]\!]_{\mathsf{config}}$. Since Ψ is not empty, the reduction is a $\rightarrow_e$-reduction and the "present" part of X is now of size i: we can apply the induction hypothesis and close the case.

Theorem 3 (Progress). *Let Ψ be a TESL$^-$ formula and ρ a satisfying run,* i.e. $\rho \in [\![\Psi]\!]_{\mathsf{TESL}}$. *For all n, there is Γ', Ψ', Φ' and a number of reductions k such that $\varnothing \models_0 \Psi \triangleright \varnothing \rightarrow^k \Gamma' \models_n \Psi' \triangleright \Phi'$ and $\rho \in [\![\Gamma' \models_n \Psi' \triangleright \Phi']\!]_{\mathsf{config}}$.*

Proof. The proof is by induction on n. For the base case, $n = 0$ we can pick $k = 0$, and both sides of the reduction are equal. For the induction step, from the induction hypothesis we have Γ', Ψ', Φ' and a number k such that $\varnothing \models_0 \Psi \triangleright \varnothing \rightarrow^k \Gamma' \models_n \Psi' \triangleright \Phi'$ and $\rho \in [\![\Gamma' \models_n \Psi' \triangleright \Phi']\!]_{\mathsf{config}}$. With Lemma 5 we obtain the required configuration at instant $n + 1$.

5 Runtime Monitoring and Testing

Our theories allow for straightforward tactic execution of the operational rules of TESL$^-$ via the Isabelle proof engine. This turned out to be too inefficient for even runs with a few simulation steps due to the internal mechanism of the proof assistant. Nevertheless, the seperate implementation of the operational semantics, named Heron, which we use for monitoring and testing [23], can be regarded with greater confidence. Indeed, its operational rules directly correspond to the operational semantics of the Isabelle/HOL implementation, which has been proved equivalent to the denotational semantics.

6 Related Work

TESL is a polychronous and polytimed language. Polymorphic time exists in the family of synchronous languages that were designed in the 1980's, such as Lustre [13], Esterel [2] and Signal [18]. In these languages, time is purely logical (there are no dates nor chronometric durations), and can be used for modeling occurrences of any kind of events, hence the polymorphic nature of time. Thereafter, Prelude [26] and Zélus [5] extended the Lustre programming language with the addition of support for metric time.

As opposed to the latter synchronous models which derive all clocks from a common root clock (defining the instants where the system reacts), polychronous models [28] do not constrain all clocks to derive from a single reaction clock, allowing a more relaxed and concurrent execution of systems. Polychrony is supported by the Signal language and in Polychronous automata [19].

Another source of inspiration in our work is CCSL [10,20], the Clock Constraints Specification Language, which supports asynchronous constraints on

the occurrence of events. It has an executable semantics [30] and a denotational semantics [9,22]. However, all these approaches do not support chronometric clocks, with dates and durations. They measure time in numbers of ticks on a clock, not in elapsed durations on a time scale. In opposition, TESL supports chronometric time, and allows different clocks to live in different time frames.

Timed automata [1] support both discrete events and measuring durations on a time scale, with several mechanization approaches of their semantics [11,27,14]. However, this time scale is global and uniform: all clocks in a timed automaton progress at the same rate. Our model considers a larger scope of time with polychronous clocks flowing independently from each other.

The GEMOC initiative [8] has been targeting the development of frameworks to facilitate the integration of heterogeneous modeling languages. In particular, the BCOoL language [29] is specifically targeted at coordination patterns for Domain Specific Events (interface of a domain specific modeling language).

7 Future work

A few directions of extension for our work are worth mentioning:

- it might be worthwhile to look for even more fundamental operators on clock-indexed models and derive a kind of core language-theory that is even more compact albeit more expressive,
- we are interested in general architectural operators allowing to combine subsystem specifications to larger ones (*e.g.*, with hidden or local clocks),
- we plan to explore the code generation features of Isabelle/HOL to produce certified solvers from the derived operational rules of our timed languages.

8 Conclusion

This study investigates the semantics of timed languages using clock-indexed Kripke models. Illustrated by a minimalist language that supports event- and timed-based constraints over polychronous clocks, we show a novel way to relate from one side, a denotational semantics, whose advantages are to be logically consistent by construction, compositional and trace-based, with an operational semantics that constructs symbolic runs and is thus suited for verification purposes. This technique is based on the observation that time, decomposed in an intuitive past-present-future pattern, can be reflected in both semantics through an operational unfolding principle. Yet, the time model we chose to study exhibits several challenging properties: time constraints can be both logical and metric, clocks are polychronous (no global clock) and polymorphic (various domain types) and clock constraints can be synchronous or asynchronous.

The unfolding principle of time in both denotational and operational semantics allows us to establish crucial properties such as stutter-invariance at the denotational level, as well as the equivalence results given by correctness and completeness. Finally, local termination and progress properties bridge the gap towards trustworthy verification tools.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**, 183–235 (1994)
2. Berry, G.: The foundations of Esterel. In: Plotkin, G., Stirling, C., Tofte, M. (eds.) Proof, Language, and Interaction, pp. 425–454. MIT Press, Cambridge, MA, USA (2000)
3. Boulanger, F., Hardebolle, C., Jacquet, C., Marcadet, D.: Semantic adaptation for models of computation. In: 2011 Eleventh International Conference on Application of Concurrency to System Design. pp. 153–162 (June 2011). https://doi.org/10.1109/ACSD.2011.17
4. Boulanger, F., Jacquet, C., Hardebolle, C., Prodan, I.: TESL: a language for reconciling heterogeneous execution traces. In: Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2014). pp. 114–123. Lausanne, Switzerland (Oct 2014)
5. Bourke, T., Pouzet, M.: Zélus: A synchronous language with odes. In: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control. p. 113–118. HSCC '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2461328.2461348, `https://doi.org/10.1145/2461328.2461348`
6. Brunette, C., Talpin, J.P., Gamatié, A., Gautier, T.: A metamodel for the design of polychronous systems. The Journal of Logic and Algebraic Programming **78**(4), 233 – 259 (2009). https://doi.org/https://doi.org/10.1016/j.jlap.2008.11.005, `http://www.sciencedirect.com/science/article/pii/S1567832608000957`, iFIP WG1.8 Workshop on Applying Concurrency Research in Industry
7. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: Abadi, M., Kremer, S. (eds.) Proceedings of the Third International Conference on Principles of Security and Trust (POST 2014). Lecture Notes in Computer Science, vol. 8414, pp. 265–284. Springer, Grenoble, France (2014)
8. Combemale, B., Cheng, B.H., France, R.B., Jezequel, J.M., Rumpe, B.: Globalizing Domain-Specific Languages, LNCS, Programming and Software Engineering, vol. 9400. Springer International Publishing (2015)
9. Deantoni, J., André, C., Gascon, R.: CCSL denotational semantics. Research Report RR-8628, Inria (Nov 2014)
10. Garcés, K., Deantoni, J., Mallet, F.: A model-based approach for reconciliation of polychronous execution traces. In: SEAA 2011 - 37th EUROMICRO Conference on Software Engineering and Advanced Applications. IEEE, Oulu, Finland (Aug 2011)
11. Garnacho, M., Bodeveix, J.P., Filali-Amine, M.: A mechanized semantic framework for real-time systems. In: Braberman, V., Fribourg, L. (eds.) Formal Modeling and Analysis of Timed Systems: 11th International Conference, FORMATS 2013, Buenos Aires, Argentina, August 29-31, 2013. Proceedings. pp. 106–120. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
12. Groote, J.F., Vaandrager, F.W.: An efficient algorithm for branching bisimulation and stuttering equivalence. In: Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP 90). Lecture Notes in Computer Science, vol. 443, pp. 626–638. Springer, Warwick University, England, UK (1990)
13. Halbwachs, N., Caspi, P., Raymond, P., Pilaud, D.: The synchronous dataflow programming language Lustre. Proceedings of the IEEE **79**(9), 1305–1320 (September 1991)

14. Hale, R., Cardell-Oliver, R., Herbert, J.: An embedding of timed transition systems in HOL. Formal Methods in System Design **3**(1), 151–174 (Aug 1993)
15. Klein, J.: Compositional Synthesis and Most General Controller. Ph.D. thesis, Technische Universität Dresden (2013)
16. Kučera, A., Strejček, J.: The stuttering principle revisited. Acta Informatica **41**(7–8), 415–434 (2005). https://doi.org/10.1007/s00236-005-0164-4
17. Lamport, L.: What good is temporal logic? In: Mason, R.E.A. (ed.) IFIP Congress on Information Processing. pp. 657–668 (1983)
18. Le Guernic, P., Benveniste, A., Bournai, P., Gautier, T.: Synchronous data flow programming with the language SIGNAL. IFAC Proceedings Volumes **20**(2), 359 – 364 (1987), 2nd IFAC Workshop on Adaptive Systems in Control and Signal Processing 1986, Lund, Sweden, 30 June-2 July 1986
19. Le Guernic, P., Gautier, T., Talpin, J.P., Besnard, L.: Polychronous automata. In: TASE 2015, 9th International Symposium on Theoretical Aspects of Software Engineering. pp. 95–102. IEEE Computer Society, Nanjing, China (Sep 2015)
20. Mallet, F., Deantoni, J., André, C., De Simone, R.: The Clock Constraint Specification Language for building timed causality models. Innovations in Systems and Software Engineering **6**(1-2), 99–106 (Mar 2010)
21. Michaud, T., Duret-Lutz, A.: Practical stutter-invariance checks for ω-regular languages. In: Fischer, B., Geldenhuys, J. (eds.) Proceedings of the 22nd International Symposium on Model Checking Software (SPIN 2015). Lecture Notes in Computer Science, vol. 9232, pp. 84–101. Springer, Stellenbosch, South Africa (2015)
22. Montin, M., Pantel, M.: Mechanizing the denotational semantics of the clock constraint specification language. In: Abdelwahed, E.H., Bellatreche, L., Golfarelli, M., Méry, D., Ordonez, C. (eds.) Model and Data Engineering. pp. 385–400. Springer International Publishing, Cham (2018)
23. Nguyen Van, H., Balabonski, T., Boulanger, F., Keller, C., Valiron, B., Wolff, B.: 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. LNCS, vol. 10419. Springer (Sep 2017)
24. Nipkow, T., Klein, G.: Concrete Semantics: With Isabelle/HOL. Springer Publishing Company, Incorporated (2014)
25. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: A Proof Assistant for Higher-order Logic. Springer-Verlag, Berlin, Heidelberg (2002)
26. Pagetti, C., Forget, J., Boniol, F., Cordovilla, M., Lesens, D.: Multi-task implementation of multi-periodic synchronous programs. Discrete Event Dynamic Systems **21**(3), 307–338 (2011), `https://hal.inria.fr/inria-00638936`
27. Paulin-Mohring, C.: Modelisation of timed automata in Coq. In: Kobayashi, N., Pierce, B.C. (eds.) Theoretical Aspects of Computer Software: 4th International Symposium, TACS 2001 Sendai, Japan, October 29–31, 2001 Proceedings. pp. 298–315. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
28. Talpin, J.P., Brandt, J., Gemünde, M., Schneider, K., Shukla, S.: Constructive polychronous systems. In: Artemov, S., Nerode, A. (eds.) Logical Foundations of Computer Science. Lecture Notes in Computer Science, vol. 7734. Springer, San Diego, CA, United States (Jan 2013)
29. Vara Larsen, M.E., Deantoni, J., Combemale, B., Mallet, F.: A behavioral coordination operator language (BCOoL). In: 18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015) (Aug 2015)
30. Zhang, M., Mallet, F.: An executable semantics of Clock Constraint Specification Language and its applications. In: Formal Techniques for Safety-Critical Systems: 4th International Workshop, FTSCS 2015. pp. 37–51. Springer, Cham (2016)