A Mechanized Semantic Framework for Real-Time Systems*

Manuel Garnacho, Jean-Paul Bodeveix, and Mamoun Filali-Amine

IRIT - CNRS - Université de Toulouse, France

Abstract. Concurrent systems consist of many components which may execute in parallel and are complex to design, to analyze, to verify, and to implement. The complexity increases if the systems have real-time constraints, which are very useful in avionic, spatial and other kind of embedded applications. In this paper we present a logical framework for defining and validating real-time formalisms as well as reasoning methods over them. For this purpose, we have implemented in the Coq proof assistant well known semantic domains for real-time systems based on labelled transitions systems and timed runs. We experiment our framework by considering the real-time CSP-based language FIACRE, which has been defined as a pivot formalism for modeling languages (AADL, SDL, ...) used in the TOPCASED project. Thus, we define an extension to the formal semantic models mentioned above that facilitates the modeling of fine-grained time constraints of FIACRE. Finally, we implement this extension in our framework and provide a proof method environment to deal with real-time system in order to achieve their formal certification.

1 Introduction

Real-time concurrent systems consist of many components which may execute in parallel and communicate data or synchronize at a specified time. Therefore, these systems are complicated to design, to analyze, to verify, and finally to implement. The complexity arises from the nondeterminism of behaviors, time constraints and the combinations of ways in which the components can interact. In order to be able to prove or verify properties on such systems one needs to formalize their semantics.

Formal semantics of specification and programming languages are *mathematical descriptions* of the meaning of programs (and their behaviors) written in these languages. They play a very important role in many areas of computer science such as *verification* of critical systems because they enable us to formally prove that these systems meet their specifications. Formal semantics of component-based languages, which are useful to design distributed or concurrent systems, are commonly defined in terms of *transition systems*. If the language has also real-time features, then its semantics can be expressed as *timed transition systems* [4] (TTS for short, see Definition 3). For this paper, we address

* This work has been partly funded by the FNRAE project Quarteft.

V. Braberman and L. Fribourg (Eds.): FORMATS 2013, LNCS 8053, pp. 106–120, 2013.

the challenge of providing a framework that allows to build *mechanized reasonings* over timed systems within a *proof assistant*. Doing so requires to *mechanize the semantics* of timed systems in adapted mathematical terms, which are here theories of transition systems and execution runs (or traces) which include time aspects.

The present work proposes also an extension to the semantic model of transition systems *with time constraints* proposed by T.Henzinger *et al* [16] that we call *time constrained transition systems* (TCTS for short, see Definition 7) since the term TTS is overloaded and already used here. The purpose of this extension is to *model more directly and easily* some real-time constructs of specification languages and then be able to prove *timed temporal properties* over critical embedded systems. We define *semantic interpretations* of systems modeled as TCTS in terms of TTS and *timed runs* (see Section 2.2) that enable to reason over these models and their time properties. Those semantic interpretations correspond to arrows 2 and 3 of Figure 1 which describes the global architecture of our framework. Arrow 1, which addresses the semantics interpretation of the syntactical constructs of our component-based language (namely FIACRE), goes beyond our current purpose and is not presented here (this is our long term goal).

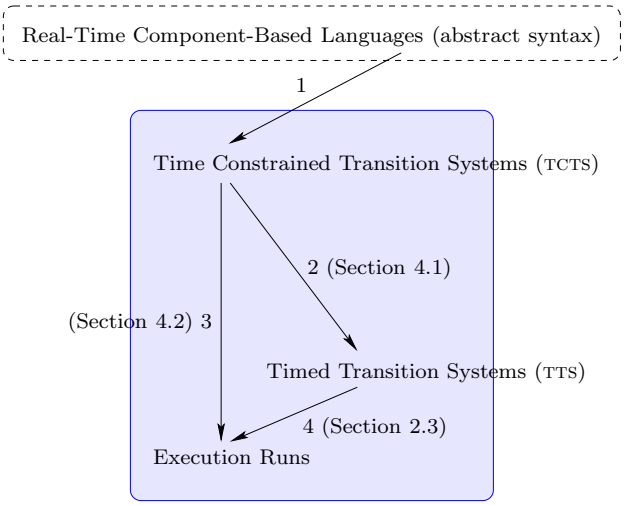

Fig. 1. Schematic description of the architecture of our semantic framework

There are several reasons for wanting to mechanize such theories based on transition systems and timed runs in an interactive theorem-prover rather than building a dedicated automatic tool [13]. The most important reason is certainly that proof assistants like Coq [7], HOL [18] or PVS [22], provide a generic and very expressive proof environment for dealing with problems which cannot be automatically decided. We claim also that the results which are encoded and checked with such tools reach probably the currently highest level of confidence

in formal software verification. In other words, proofs built and then checked with, for example Coq, avoid to a very high extent fallacious and incomplete arguments that are so easy and so common to find in standard mathematical proofs. Then, all the proofs that we provide in this paper have been encoded in the calculus of inductive constructions (CIC for short) using the Coq proof assistant, and then automatically checked to guarantee the mathematical soundness of the proofs [1]. However, although all of this formalization work (shaded part of Figure 1) has been carried out in Coq, we refrained from displaying Coq syntax here. We adopt standard mathematical notations as much as possible for the sake of clarity and because a large part of this work is about semantics of real-time systems that is widely independent of Coq.

Overview. The remainder of this paper is structured as follows: Section 1.1 introduces the general principles of component-based languages. We describe their timed semantics through a toy example. Section 2 recalls the basis of transition systems and execution runs. Section 3 defines an extension of transition systems with time constraints. Section 4 is dedicated to the semantic interpretations of transition systems in two distinct models that address the dynamic time aspects of real-time systems. Then, in Section 5 we explain why our work has been implemented in the interactive theorem prover Coq and how it can be used for formal verification. We draw a conclusion and present some directions for future works in Section 6.

1.1 Timed Semantics of Component-Based Languages

Existing specification languages inspired by [17] are useful to represent both behavioral and timing aspects of concurrent (or distributed) real-time systems. For example, FIACRE [6] is defined over two main notions : (1) *processes* that describe the behavior of sequential components. A process is defined by a set of control states (or positions), each associated with a piece of program built from deterministic constructs available in classical programming languages, nondeterministic constructs (nondeterministic choice of the transition to take, for example), communication events on ports, and jumps to next position; (2) *components* that describe the composition of processes. A component is defined as a parallel composition of components and/or processes communicating through *ports* and shared variables.

In this paper we focus only on nondeterministic transitions and timing constraints of synchronizations (composition and priorities are adressed in [12]). FIACRE allows to model real-time behaviors by using timed ports, *i.e.* ports with their own timer (or clock). A timed transition connects two control states using such a port. Moreover, FIACRE allows to code such transitions in two ways : the first one preserves the clocks of other *enabled transitions* whereas the second one resets the clock of other *enabled transitions*.

[1] We invite the interested reader to browse the sources of our development at the web address : `http://www.irit.fr/~Manuel.Garnacho/Coq/FORMATS13`.

The example presented in Figure 2 describes a process which declares three local ports, `p`, `q` and `r`, and two variables of type `nat`, namely `x` and `y`. In this specification language, local ports are only used to temporize transitions and processes which use those ports do not get "synchronized" with others through those ports. The port `p` is time constrained by the interval $[1, 3[$, `q` by $]3, \infty[$ and `r` by $[1, 1]$. Considering `p`, it means that when a transition which synchronizes on it is enabled, it must fire after waiting *at least* one (included) and *at most* three (not included) time units. In the same way, `q` delays the process `P` for at least three (not included) time units [2] and `r` exactly one unit of time.

Then, we see that in the process `P`, there are two control states, namely `s0` and `s1`. Only `s0` is an initial one, which means that the process has to start its execution from it. But let us first take a look at `s1` : *from* this control state, a *nondeterministic choice* is defined with the construction `select`. This choice includes two concurrent transitions : the first one that synchronizes on `p` then adds to `y` the value of `x`; the second one that synchronizes on `q` then assigns to `x` the value of `y` plus 2. Afterward, each of those transitions *jumps* to a next control state with the construction `to`, *i.e.* it jumps back to `s1` in the case of an action on `p` and goes to `s0` in the other case. The timed-semantics of the construction `to`, consists in *resetting* the timer of all transitions included in the same nondeterministic choice (we provide in comments the set of ports that *have to be reset* for each transition). This notion will be formalized in Section 3, thanks to an extension of the semantic model introduced in [16]. Consequently, from the control state `s1` the process `P` is never synchronized on `q`, so the variable `x` will never be assigned and it never goes back to `s0`. Indeed, after every single unit of time the process will be synchronized on `p` before strictly two more time units, after which it resets the timer of `q` before having waited enough time to synchronize on it. Thus, the second transition of the nondeterministic choice from `s1` of `P` is *dead code* due to the time constraints of `P`.

Now, through the same example we present another construction that *loops* to the current control state but with a different timed-semantics. This construction is still related to the nondeterministic choice of transitions and is defined with the keyword `loop`. It consists in going back to the current control state while *preserving the time already waited by other transitions* (which were not taken) included in the same nondeterministic choice as the one that has been taken. The clock of the taken transition is reset however. For example, in the nondeterministic choice from the control state `s0` of `P`, the first transition loops after synchronizing on `r` and then increments `x`. This transition occurs after exactly every single time unit and then only resets its own timer. On the other hand, the nondeterministic choice allows the process to synchronize on `q` at any time after being continuously enabled for at least three units of time. Note that the transition through `q` from `s0` may never occur because of its unconstrained maximal delay. Proving such semantics properties on the value of `x` and `y` or that the transition through `q` from `s1` is *dead code*, can be as much crucial as difficult.

[2] Time bounds are either integral or infinite. Elapsing of time is continuous.

```
process P is

  ports p in [1,3[, q in ]3,∞[, r in [1,1]
  var x, y : nat := 0
  states s0 s1
  init to s0

  from s0 select
 /*1*/     r ; x := x+1; loop    /* reset = {r} */
 /*2*/  [] q ; to s1             /* reset = {r,q} */
      end

  from s1 select
 /*3*/     p ; y := x+y; to s1   /* reset = {p,q} */
 /*4*/  [] q ; x := y+2; to s0   /* reset = {p,q} */
      end
```

Fig. 2. An example real-time process specified in FIACRE

Related Work. Starting with the work of T. Henzinger *et al.* [16, 15], we extend their mathematical semantic model for specifying real-time processes such as the one presented in Figure 2. All definitions and theorems presented in the following are directly derived from a mechanization developed in Coq. This part of our contributions connects to the previous work of R. Cardell-Oliver *et al.* [14] about the embedding in HOL of [16] and C. Paulin-Mohring about the formalization of timed automata in Coq [20]. With respect to the Petri Net community, our work is in the spirit of B. Berard *et al.* [4]. However, we are concerned by a component-based language with states and ports and we go further since we adress its mechanization. Morever, our work is not in the same scope as [13] because our goal is not to develop an automatic verification tool certified in Coq. Our purpose is to provide a real-time semantic framework which can be used to formalize real-time patterns and more generally real-time specification languages. As a matter of fact, we present in the following the formalization of the FIACRE language where we are especially interested in the semantic differences between the contructs `loop` and `to` (see Section 1.1). Also, the proof environment of our framework allows to *certify transformations* between real-time formalisms. For instance in [8], we are concerned by the transformation from AADL to FIACRE, which requires formal certification for avionic or spatial applications.

2 Semantics Kernel

This section is dedicated to the basis of classical semantic notions we use in our work. These notions are of two kinds: *timed transition systems* and *timed execution runs* [4]. Moreover, *both include time* and since *time is dense* in most real-time specification languages (including FIACRE), we formalize time in this paper with non-negative real numbers : $\mathbb{R}^+$. However, in our Coq mechanization we have developed an axiomatic *theory of time*[3] which can be instantiated by constructive reals [13] as well as by `nat` to encode discrete time (if necessary).

[3] `http://www.irit.fr/~Manuel.Garnacho/Coq/FORMATS13/time.html`.

These semantic models enable us to describe formally the dynamics of real-time systems, mixing the evolution of their states and the progress of time, to reason over those with rigorous logical methods.

2.1 Transition Systems

First of all, we remind the basic model of *labelled transition systems* [2] that is usually used to give a mathematical representation of computer programs.

Definition 1 (Labelled Transition Systems). *A labelled transition system (*LTS *for short) is a* 4*-tuple* $lts \stackrel{def}{=} \langle L, S, \boldsymbol{init}, \boldsymbol{next}\rangle$*, where :* L *is the set of* labels *of* lts*;* S *is a set of* states *(or stores);* $\boldsymbol{init}$ *is a non-empty subset of* S *that defines the initial states of* lts*; and* $\boldsymbol{next}$ *defines the set of transitions of* lts *that are triplets of the form* $(sto, \ell, sto') \in S \times L \times S$*, also denoted as* $sto \xrightarrow{\ell} sto'$*, where* $sto \in S$ *is the source state,* $\ell \in L$ *is the label and* $sto' \in S$ *the target state of the transition.*

Definition 2. *Assuming a* LTS*,* $lts \stackrel{def}{=} \langle L, S, \boldsymbol{init}, \boldsymbol{next}\rangle$*, a label* $\ell \in L$ *is* enabled *from a state* $sto \in S$*, if there exists a state* $sto' \in S$ *such that the triplet* (sto, ℓ, sto') *belongs to* $\boldsymbol{next}$*. Formally, we define the predicate* $\boldsymbol{enb}$ *over* $S \times L$ *as* $\boldsymbol{enb}_{sto}(\ell) \stackrel{def}{=} \exists sto' \in S, \boldsymbol{next}(sto, \ell, sto')$

Definition 3 (Timed Transition Systems). *A timed transition system (*TTS *for short) over a set of labels* L*, is a* LTS *over* $L \cup \mathbb{R}^+$*,* $\langle L \cup \mathbb{R}^+, S, \boldsymbol{init}, \boldsymbol{t\_next}\rangle$*, that satisfies the following properties :*

- *zero delay :* $\forall sto \in S, sto \xrightarrow{0} sto$
- *time-determinism :* $\forall sto, sto', sto'' \in S, \forall \delta \in \mathbb{R}^+,$
$$sto \xrightarrow{\delta} sto' \Rightarrow sto \xrightarrow{\delta} sto'' \Rightarrow sto' = sto''$$
- *additivity :* $\forall sto, sto', sto'' \in S, \forall \delta, \delta' \in \mathbb{R}^+,$
$$sto \xrightarrow{\delta} sto' \Rightarrow sto' \xrightarrow{\delta'} sto'' \Rightarrow sto \xrightarrow{\delta+\delta'} sto''$$
- *continuity :* $\forall sto, sto'' \in S, \forall \delta', \delta'' \in \mathbb{R}^+,$
$$sto \xrightarrow{\delta'+\delta''} sto'' \Rightarrow \exists sto', sto \xrightarrow{\delta'} sto' \wedge sto' \xrightarrow{\delta''} sto''$$

2.2 Executions

We remind now the classical representation of runs in a timed context, that is useful to represent *the behaviors* of systems and then reason with temporal linear logic [21] about those.

Definition 4 (Timed execution states). *With respect to a set of states* S *and a set of events* L*, a* timed execution state *(*TES *for short),* s*, is a triplet of the form* $\langle sto, evt, now\rangle$ *where :*

- $sto_s \in S$*, is the state (or store) reached in* s
- $evt_s \in L \cup \{start\} \cup \{div\}$*, is the event that has reached* s*.*
- $now_s \in \mathbb{R}^+$ *gives the current global time (or date) in* s*.*

Remarks. The two special events *start* and *div* are seen, respectively, at the *beginning* of a run and whenever *time is diverging* (in case of deadlock, for instance) in a run.

Definition 5 (Runs). *A* run *(or execution sequence) is a function σ, from $\mathbb{N}$ to* TES*, that represents a (possibly finite) sequence of events all of which reach a state at a given date. For a* TES *s reached at a position i of σ, we have $\sigma_i = s$.*

Remarks. For the sake of conciseness, we allow ourselves to substitute s (σ_i) by i in the following whenever it might simplify the notations (as below). Also, according to our representation of runs, finite ones are encoded using a repetition of the special event *div* after the last state.

Definition 6 (Well formed runs). *According to the previous definition, a run σ is* well formed *if it satisfies the following properties :* **start** *: $\mathsf{evt}_0 = start \wedge \mathsf{now}_0 = \delta^0_\sigma$; and* **monotonicity** *: $\forall i \in \mathbb{N},\ \mathsf{now}_i \leq \mathsf{now}_{i+1}$, where δ^0_σ is the initial delay associated to the beginning (the event start) of the run σ.*

Remark. This definition of timed runs does not satisfy the *progress* property ($\forall r \in \mathbb{R}^+,\ \exists i \in \mathbb{N},\ r \leq \mathsf{now}_i$) [16] because of *Zeno behaviors* (that include an infinite number of discrete steps in a finite amount of time) that may be generated from the component-based source language, as for instance in FIACRE, that we want to capture.

2.3 Semantics of Timed Transition Systems in Terms of Runs

We define now the semantics of TTS in terms of runs (arrow 4 of Figure 1).

Predicates. In order to describe the semantics of TTS*s* in terms of runs, we have to define two predicates that express when a given label may be taken and when a label is taken. Assume a given run σ at some position i, σ-$\boldsymbol{enb}_i(\ell)$ means that the label ℓ may be taken at i and $\boldsymbol{taken}_i(\ell)$ means that ℓ is taken at i. Formally :

- σ-$\boldsymbol{enb}_i(\ell) \stackrel{def}{=} \exists \mathsf{sto}, \mathsf{sto}' \in S, \exists \delta \in \mathbb{R}^+, \boldsymbol{next}(\mathsf{sto}_i, \ell, \mathsf{sto}) \wedge \boldsymbol{next}(\mathsf{sto}, \delta, \mathsf{sto}')$
- $\boldsymbol{taken}_i(\ell) \stackrel{def}{=} \mathsf{evt}_{i+1} = \ell$ (sto_{i+1} corresponds to one of the sto' above)

Remark. The intermediate state, sto, used in the definition of σ-$\boldsymbol{enb}$ is needed because the runs that we consider do not include explicit *duration*. There are other formalisms of execution runs where time passing is explicit (mixing durations and events) but here, to avoid *Zeno phenomena* (coming from the TTS*s* and not from the systems theirselves) in runs, we only consider events.

Semantic Interpretation in Terms of Runs. The run-semantics of a TTS, $tts \stackrel{def}{=} \langle L \cup \mathbb{R}^+, S, \boldsymbol{init}, \boldsymbol{t\_next} \rangle$, is a set of well formed runs $\Sigma(tts)$, that expresses all possible behaviors of *tts* at execution, such as each of those has to fulfill that $\boldsymbol{init}(\mathsf{sto}_0)$ and the following step condition :

$\forall i$, either

$\exists \ell$ such that $\exists \mathsf{sto} \in S, \exists \delta \in \mathbb{R}^+, \boldsymbol{t\_next}(\mathsf{sto}_i, \ell, \mathsf{sto}) \wedge \boldsymbol{t\_next}(\mathsf{sto}, \delta, \mathsf{sto}_{i+1})$

and $\mathsf{now}_{i+1} = \mathsf{now}_i + \delta$ and $\boldsymbol{taken}_i(\ell)$ (a label is taken)
or $\neg\exists \ell$ such that $\sigma\text{-}\boldsymbol{enb}_i(\ell)$ and $\forall j > i, \mathsf{sto}_j = \mathsf{sto}_i \ \wedge \ \mathsf{evt}_j = div$ (time is diverging).

Graphically, when a label ℓ is taken at some position i, it can be represented as $\sigma_i \xrightarrow{\ell} s \xrightarrow{\delta} \sigma_{i+1}$, where s is an implicit intermediate TES which is not visible in the run, defined as $\langle \mathsf{sto}_{i+1}, \mathsf{evt}_{i+1}, \mathsf{now}_i \rangle$.

3 A Time Constrained Model for Real-Time Systems

We give in this section an *extended definition* of the semantics model proposed by T. Henzinger *et al.* in [16] that deals with advanced (*i.e.* fine-grained) timed constructs such as the ones presented in Section 1.1. As in [16], we incorporate real-time constraints to LTS in order to ensure that a labelled transition is fired neither too early nor too late. An extension to their initial model is the so-called *reset relation* that specifies which clocks (or timers) are reset after the firing of a given transition. We call this model *Time Constrained Transition Systems* (TCTS for short) since time features are only expressed as syntactic timed constraints on transitions. We have introduced this semantics model in order to verify real-time processes such as the one presented in Figure 2 more easily. Moreover, the exhibited semantic model is interesting by itself since it can be reused to define the semantics of real-time component-based langages such as FIACRE [11] or BIP[3] or even high-level specification models such as Timed Petri Nets with *read-arcs* [4]. We also present two timed-semantic interpretations, corresponding to arrows 2 and 3 of Figure 1.

3.1 Time Constrained Transition Systems

In order to reason about timing of transitions, we identify each transition by its name. For instance, according to the process P of Figure 2, its corresponding TCTS include four names, one for each of the four possible transitions from its two nondeterministic choices, even if the number 2 and 4 synchronize both on q. Moreover, we suppose that there is an implicit clock (while time is explicit in TTS) associated to every transition. Then, this model allows to specify static time constraints over the transitions of a given system.

Definition 7. *A Time Constrained Transition System, namely tcts, is a* 8*-tuple* $\langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I \rangle$*, where* $\langle L, S, \boldsymbol{init}, \boldsymbol{next} \rangle$ *is a* LTS*, and :*

- T *is the set of* (the names of) transitions *of tcts.*
- lbl *is a function from* T *to* L*, that associates to every transition a label.*
- $\mathcal{R}$ *is the reset transition relation.* $(tr, tr') \in \mathcal{R}$ *states that at* execution*, the firing of* tr *resets the* implicit clock *of* tr'*. Otherwise, the implicit clock associated to* tr' *keeps running. For all* $tr \in T$*,* $(tr, tr) \in \mathcal{R}$ *(*$\mathcal{R}$ *is refelexive).*
- I *is a function that assigns to every label* $\ell \in L$ *a non-empty interval of* $\mathbb{R}^+$*.* I_ℓ *(or* $I(\ell)$*) specifies both minimal (lower) and maximal delay (upper bound) to elapse once* ℓ *has been enabled (see Definition 2).*

We introduce the downward and the upward closure of an interval I_ℓ defined respectively by $\overleftarrow{I_\ell} \stackrel{def}{=} \{t \mid \exists r \in I_\ell, t \leq r\}$ and $\overrightarrow{I_\ell} \stackrel{def}{=} \{t \mid \exists r \in I_\ell, r \leq t\}$. For example, if I_ℓ is defined by the interval $[min, max]$ then $\overleftarrow{I_\ell}$ is $[0, max]$ and $\overrightarrow{I_\ell}$ is $[min, \infty[$.

Example 1. Consider the process given in Figure 2, we define its corresponding TCTS as $\langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$, where :

- $L \stackrel{def}{=} \{\ell_p, \ell_q, \ell_r\}$
- $S \stackrel{def}{=} \{(x, y, loc) \mid x, y \in \mathbb{N} \wedge loc \in \{l_0, l_1\}\}$
- $\forall \mathsf{sto} \in S, \boldsymbol{init}(\mathsf{sto}) \stackrel{def}{=} \mathsf{sto}(x) = 0 \wedge \mathsf{sto}(y) = 0 \wedge \mathsf{sto}(loc) = l_0$
- $\boldsymbol{next}(\mathsf{sto}, \ell, \mathsf{sto}') \stackrel{def}{=}$

$$\bigwedge \begin{pmatrix} \mathsf{sto}(loc) = l_0 \Rightarrow \bigvee \begin{pmatrix} \ell = \ell_r \wedge \mathsf{sto}' = \mathsf{sto}[\mathsf{sto}(x) + 1/x] \\ \ell = \ell_q \wedge \mathsf{sto}' = \mathsf{sto}[l_1/loc] \end{pmatrix} \\ \mathsf{sto}(loc) = l_1 \Rightarrow \bigvee \begin{pmatrix} \ell = \ell_p \wedge \mathsf{sto}' = \mathsf{sto}[\mathsf{sto}(x) + \mathsf{sto}(y)/y] \\ \ell = \ell_q \wedge \mathsf{sto}' = \mathsf{sto}[\mathsf{sto}(y) + 2/x][l_0/loc] \end{pmatrix} \end{pmatrix}$$

- $T \stackrel{def}{=} \{tr_1, tr_2, tr_3, tr_4\}$
- $\mathsf{lbl}(tr_1) \stackrel{def}{=} \ell_r \mid \mathsf{lbl}(tr_2) \stackrel{def}{=} \ell_q \mid \mathsf{lbl}(tr_3) \stackrel{def}{=} \ell_p \mid \mathsf{lbl}(tr_4) \stackrel{def}{=} \ell_q$
- $\mathcal{R} \stackrel{def}{=} \{(tr_i, tr_i) \mid 1 \leq i \leq 4\} \cup \{(tr_2, tr_1), (tr_3, tr_4), (tr_4, tr_3)\}$
- $I_{\ell_p} \stackrel{def}{=} [1, 3[, I_{\ell_q} \stackrel{def}{=}]3, \infty[$ and $I_{\ell_r} \stackrel{def}{=} [1, 1]$.

Remarks. Because of the functional relation from T to L, we consider (by abuse of notation) in the remaining that, for any TCTS, $\boldsymbol{next}$ is also defined over $S \times T \times S$. Furthermore, assuming a TCTS, $tcts \stackrel{def}{=} \langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$, a transition $tr \in T$ is *enabled* from a state $\mathsf{sto} \in S$, if there is a state $\mathsf{sto}' \in S$ such that the triplet $(\mathsf{sto}, tr, \mathsf{sto}')$ belongs to $\boldsymbol{next}$. Formally, we extend the predicate $\boldsymbol{enb}$ over $S \times T$ as $\boldsymbol{enb}_{\mathsf{sto}}(tr) \stackrel{def}{=} \exists \mathsf{sto}' \in S, \boldsymbol{next}(\mathsf{sto}, tr, \mathsf{sto}')$. Also, considering a transition $tr \in T$, we write I_{tr} instead of $I_{\mathsf{lbl}(tr)}$ in order to simplify notations.

As we have seen, TCTS is a mathematical model that allows to specify *time aspects* of real-time systems. We have also defined the composition of TCTS to model concurrent or distributed aspects of systems in the same formalism (see [12]). The semantics of this composition is based on the composition of LTS. However, due to lack of space, we choose to focus on the real-time aspects here.

4 Semantic Interpretations

We consider in this section two models to interpret TCTS w.r.t. time semantic aspects. The first one is TTS with explicit time steps together with time assumptions (see Definition 3). This model allows to reason over behaviors of a given TCTS using a branching time logic as [10] or (bi-)simulation relations [4].

Secondly, TCTS timing aspects are also interpreted as sets of *runs*, that describe the executions of a TCTS as sequences of timed events. This model allows to define and to prove the satisfaction of temporal properties expressed in a linear temporal logic [21] by a given TCTS.

4.1 Semantics of TCTSs in Terms of TTSs

Now, we want to give the semantics of TCTS in terms of TTS, interpreting time constraints by timed transitions. The purpose, among others, is to be able to reason on TCTS*s* at the TTS level.

So, given a TCTS, $tcts \stackrel{def}{=} \langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$, we define a corresponding TTS through the semantics function $[\![\ ]\!] : \text{TCTS} \rightarrow \text{TTS}$, such that $[\![tcts]\!] = \langle L \cup \mathbb{R}^+, \mathcal{S}, \boldsymbol{init}, \boldsymbol{t\_next}\rangle \in \text{TTS}$, where :

- a state of $s \in \mathcal{S}$ is a pair over $S \times (T \rightarrow \mathbb{R}^+)$, such that $s \stackrel{def}{=} \langle \mathsf{sto}, \mathsf{clk}\rangle$ with :
  - $\mathsf{sto}_s \in S$ is a store of the underlying TCTS.
  - $\mathsf{clk}_s : T \rightarrow \mathbb{R}^+$ associates to every transition $tr \in T$ an *explicit clock* that is needed to count elapsed time since tr is enabled.
  - and for every $tr \in T$, if $\boldsymbol{enb}_{\mathsf{sto}_s}(tr)$ then $\mathsf{clk}_s(tr) \in \overleftarrow{I_{tr}}$
- $\boldsymbol{t\_next}$ is a predicate over $\mathcal{S} \times (L \cup \mathbb{R}^+) \times \mathcal{S}$ defined as :
  $\forall s \in \mathcal{S}, \forall s' \in \mathcal{S}, \forall \ell \in L \cup \mathbb{R}^+,\ \bigwedge \begin{pmatrix} \ell \in L \Rightarrow \boldsymbol{discrete}(s, \ell, s') \\ \ell \in \mathbb{R}^+ \Rightarrow \boldsymbol{delay}(s, \ell, s') \end{pmatrix}$, with :

  - $\forall s \in \mathcal{S}, \forall s' \in \mathcal{S}, \forall \ell \in L,\ \boldsymbol{discrete}(s, \ell, s') \stackrel{def}{=} \exists tr \in T, \mathsf{lbl}(tr) = \ell$ and
    $\boldsymbol{next}(\mathsf{sto}_s, tr, \mathsf{sto}_{s'})$ and $\mathsf{clk}_s(tr) \in \overrightarrow{I_{tr}}$
    and $\forall tr', \mathsf{clk}_{s'}(tr') = \begin{cases} 0 \text{ if } \neg \boldsymbol{enb}_{\mathsf{sto}_s}(tr') \vee (tr, tr') \in \mathcal{R} \\ \mathsf{clk}_s(tr') \text{ else} \end{cases}$

    In other words, considering two timed states s and s' of $[\![tcts]\!]$, the label of a discrete transition tr links them if (1) tr is a transition between sto_s and $\mathsf{sto}_{s'}$ in the source TCTS; (2) since enabled, tr is delayed enough time units (according to the specification I_{tr}); and (3) for every other enabled transition tr', its clock is preserved if and only if tr does not reset it (clocks are set to 0 for all other transitions, tr included since $(tr, tr) \in \mathcal{R}$).

  - $\forall s \in \mathcal{S}, \forall s' \in \mathcal{S}, \forall \delta \in \mathbb{R}^+,\ \boldsymbol{delay}(s, \delta, s') \stackrel{def}{=} \forall tr \in T,$
    $\mathsf{clk}_{s'}(tr) = \mathsf{clk}_s(tr) + \delta$
    and if $\boldsymbol{enb}_{\mathsf{sto}_s}(tr)$ then we must have $\mathsf{clk}_s(tr) + \delta \in \overleftarrow{I_{tr}}$
    and $\mathsf{sto}_{s'} = \mathsf{sto}_s$

    In other words, considering two timed states s and s' of $[\![tcts]\!]$, time passes (δ units) between them if for all transitions tr of $tcts$, (1) its clock goes δ time units from s to s' (2) after being enabled δ more units time, the maximal delay to take tr is not reached; and (3) stores are the same.

Theorem 1. $\forall tcts : \text{TCTS}, [\![tcts]\!] : \text{TTS}$.

The proof consists in proving for any $tcts \in \text{TCTS}$, that its semantic interpretation, $[\![tcts]\!]$, satisfies the four properties of TTS*s* (see Definition 3).

4.2 Semantics of TCTSs in Terms of Timed Runs

The behavior of a TCTS can be expressed by a set of *runs* (see Definition 5) in order to reason about real-time systems using temporal logics. Here, according to the definition of TCTS with the set $\mathcal{R}$, we need to extend TES*s* of runs in order to deal with resetting of clocks. Runs of a TCTS are now sequences of TES*s* as in Definition 8, but with an additional predicate on the names of transitions.

Definition 8. *Assuming a* TCTS, $tcts \stackrel{def}{=} \langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$, *a* timed execution state *of tcts, s, is now a* 4*-tuple* $\langle \mathsf{sto}, \mathsf{evt}, \mathsf{now}, \sigma\text{-}\boldsymbol{reset}\rangle$ *where :*

- $\mathsf{sto}_s \in S$, *is the state (or store) reached in s*
- $\mathsf{evt}_s \in L \cup \{start\} \cup \{div\}$, *is the event by which s is attained.*
- $\mathsf{now}_s \in \mathbb{R}^+$ *gives the global time in s.*
- $\sigma\text{-}\boldsymbol{reset}_s$ *is a function from T to* $\mathbb{B}$ *(or a predicate over T) that indicates which transitions have had their* implicit clock *reset when s has been reached.*

Predicates. Assume a given run σ at some position i, $\sigma\text{-}\boldsymbol{enb}_i(tr)$ means that the transition $tr \in T$ may be taken at i and $\boldsymbol{taken}_i(tr)$ means that tr is taken at i. Formally :

- $\sigma\text{-}\boldsymbol{enb}_i(tr) \stackrel{def}{=} \exists \mathsf{sto} \in S, \boldsymbol{next}(\mathsf{sto}_i, tr, \mathsf{sto})$
- $\boldsymbol{taken}_i(tr) \stackrel{def}{=} \bigwedge \begin{pmatrix} \mathsf{evt}_{i+1} = \mathsf{lbl}(tr) \\ \boldsymbol{next}(\mathsf{sto}_i, tr, \mathsf{sto}_{i+1}) \\ \forall tr', \sigma\text{-}\boldsymbol{reset}_{i+1}(tr') \Leftrightarrow (tr, tr') \in \mathcal{R} \end{pmatrix}$

Remark. Here, the predicate $\boldsymbol{taken}()$ is more sophisticated than in Section 2.3 because it deals with T and no more with L. Indeed, the first condition of the conjunction above is not enough because two transitions of T might satisfy it.

Run-Semantics of a TCTS. Now, we are able to express the semantic interpretation of a TCTS in terms of runs, as we did in Section 2.3 with TTS. Consider a TCTS, $tcts \stackrel{def}{=} \langle L, T, S, \mathsf{lbl}, \boldsymbol{init}, \boldsymbol{next}, \mathcal{R}, I\rangle$, we define below the set of well formed runs that we call $\Sigma(tcts)$. Every run σ of $\Sigma(tcts)$, which expresses a behavior of $tcts$, has to fulfill the four following conditions :

Initial timed execution state condition : $\boldsymbol{init}(\mathsf{sto}_0) \wedge \mathsf{now}_0 \in \overrightarrow{I_{\mathsf{evt}_1}}$

In other words, any run must start from an initial state of its underlying TCTS and the taken transition at position 0 has been delayed enough.

Next timed execution state condition :

$\forall i \in \mathbb{N}$, either $\exists tr \in T$ such that $\boldsymbol{taken}_i(tr)$ (a discrete transition is taken)
or $\neg\exists tr \in T$ such that $\sigma\text{-}\boldsymbol{enb}_i(tr)$ and $\mathsf{sto}_i = \mathsf{sto}_{i+1} = \mathsf{sto}_{i+2} = ... \wedge$
$\sigma_i \xrightarrow{div} \sigma_{i+1} \xrightarrow{div} ...$ (time diverging)

In other words, a next timed execution state can be reached from a state at position i of σ if either a discrete transition can be taken or a deadlock is met and time goes to infinity (the tag event *div* was introduced especially for this purpose in order to distinguish deadlock from the identity transition).

Lower bound condition : $\forall tr \in T$, and $\forall j \in \mathbb{N}$, if $\boldsymbol{taken}_j(tr)$ then
$\mathsf{now}_j \in \overrightarrow{I_{tr}}$ and
$\forall i \in \mathbb{N}$ such that $i < j$,
if $\mathsf{now}_j - \mathsf{now}_i \notin \overrightarrow{I_{tr}}$ then $\sigma\text{-}\boldsymbol{enb}_i(tr) \wedge \neg\sigma\text{-}\boldsymbol{reset}_{i+1}(tr)$

In other words, a transition tr can be taken only after being continuously enabled for enough (means that it is delayed for at least the lower bound of I_{tr}) time units without being reset by the execution of another transition. The condition $\mathsf{now}_j \in \overrightarrow{I_{tr}}$ is required in the case where tr is enabled since the start of the execution. Indeed, in this case the premise $\mathsf{now}_j - \mathsf{now}_i \notin \overrightarrow{I_{tr}}$ is useless and we need to know that tr has been continuously enabled even before the position σ_0. It seems that this case has been omitted in [16].

Upper bound condition : $\forall tr \in T$ and $\forall i \in \mathbb{N}$,
$\exists j \in \mathbb{N}$ such that $i \leq j$ and $\mathsf{now}_j - \mathsf{now}_i \in \overleftarrow{I_{tr}}$
and if $\sigma\text{-}\boldsymbol{enb}_j(tr)$ then $\boldsymbol{taken}_j(tr) \vee \sigma\text{-}\boldsymbol{reset}_j(tr)$

In other words, once enabled, a transition tr is not delayed for too long or it has been reset meanwhile. A transition cannot be continuously enabled for more than the upper bound of I_{tr} without being taken or reset.

4.3 Soundness of the Semantic Interpretations

Now we want to establish the soundness of our semantic interpretations by proving that the runs built straight from a given TCTS are the same that those built from its semantic interpretation in terms of a TTS. To do so, we must first define a semantic interpretation from timed execution state associated to TCTS*s* to the ones associated to their corresponding TTS*s*. So, we introduce the function γ from $\Sigma(tcts)$ to $\Sigma([\![tcts]\!])$, where states are in $\mathcal{S} \stackrel{def}{=} S \times (T \to \mathbb{R}^+)$ (see Section 4.1), *i.e.* every state associates a clock to every transition.

Definition 9. $\forall tcts : \text{TCTS}, \forall \sigma \in \Sigma(tcts), \forall i \in \mathbb{N},\ \gamma(\sigma_i) \stackrel{def}{=}$

$$\begin{cases} \langle\langle \mathsf{sto}_0, (\lambda tr \in T.\ \delta^0_\sigma)\rangle, start, \mathsf{now}_0\rangle \ \textit{if } i = 0, \\ \langle \left\langle \mathsf{sto}_i,\ \lambda tr \in T. \begin{pmatrix} \delta \ \textit{if } \neg\sigma\text{-}\boldsymbol{enb}_{i-1}(tr) \ \textit{or } \sigma\text{-}\boldsymbol{reset}_i(tr) \\ \mathsf{clk}_{\gamma(\sigma_{i-1})}(tr) + \delta \ \textit{otherwise} \end{pmatrix} \right\rangle, \mathsf{evt}_i, \mathsf{now}_i\rangle \ \textit{else} \end{cases}$$

where $\delta \stackrel{def}{=} (\mathsf{now}_i - \mathsf{now}_{i-1})$ *and* δ^0_σ *is the initial delay of* σ*, which is* now_0*.*

Then, we define the semantics interpretation of runs of TCTS*s in terms of runs of corresponding* TTS*s as* $\forall tcts \in \text{TCTS}, \forall \sigma \in \Sigma(tcts),\ \gamma(\sigma) \stackrel{def}{=} \forall i \in \mathbb{N}, \gamma(\sigma_i)$.

Theorem 2. $\forall tcts : \text{TCTS}, \forall \sigma : \mathbb{N} \to \text{TES},\ \sigma \in \Sigma(tcts) \Leftrightarrow \gamma(\sigma) \in \Sigma([\![tcts]\!])$.

5 Mechanizing and Reasoning with a Proof Assistant

We explain in this section *why* we have implemented during the last two years the formalization work presented in this paper in an interactive theorem prover

based on type-theory. First of all, the purpose is to validate with a very high level of confidence all the theorems about semantic interpretations of TCTS that we have presented previously. Moreover, using formal fundational proofs can also be useful to go beyond model-checking techniques and capabilities [13], when it is needed for example to check parameterized systems. To make that practical, we have *deeply embedded* in Coq a *timed linear temporal logic*[4] (called timed-SELTL) which is an extension of the State/Event linear temporal logic (SE-LTL) [9] since it embeds time intervals within temporal operators, together with a *proof methodology* dedicated to the *formal certification* of real-time systems.

Definition 10. *A timed-SELTL formula f is defined inductively as follows (where φ_S is a predicate over S, e is label of L and I is a time interval) :*

$$f ::= \varphi_S \mid e \mid \dot{\neg} f \mid f \mathbin{\dot{\wedge}} f \mid f \; \mathcal{U}_I f$$

Remarks. As usual, we define the three temporal operators $\Diamond$ (eventually), $\Box$ (always) and $\mathcal{W}$ (weak until) as $\Diamond_I f \stackrel{def}{=} \top \; \mathcal{U}_I f$, $\Box_I f \stackrel{def}{=} \neg(\top \; \mathcal{U}_I \neg f)$ and $f_1 \; \mathcal{W}_I f_2 \stackrel{def}{=} (f_1 \; \mathcal{U}_I f_2) \mathbin{\dot{\vee}} (\Box_{\overleftarrow{I}} f_1)$, where $\mathcal{U}_I$ is the *until operator* constrained with a time interval. We introduce as well the connectors $\dot{\Rightarrow}$ and $\dot{\vee}$. The *semantics* of this logic is defined over our formalization of runs and its implementation is available in our Coq framework.

Then, in order to reason and prove the correctness of real-time specifications modelled as TCTS*s*, we have *embedded* in our Coq framework a proof methodology very close to the one proposed in [15], based on the *same class of real-time properties* they consider (bounded response, bounded unless and bounded invariance properties) expressed as timed-SELTL formulæ. To do so, we have established *several* Coq *theorems* such as the two given below, where p, q, r and Φ are predicates over S. These theorems provide efficient proof rules since they enable us to certify in Coq real-time specifications expressed as timed-SELTL formulæ *over runs* (their conclusion), but *reasoning locally* (thanks to their premises) on TCTS*s* with Hoare triple and logical formulae. We have also proved *transitivity* and *disjunction* lemmas which are essential to make the method usable.

$$\frac{\begin{array}{l}(1)\ \forall s, p(s) \Rightarrow \neg \boldsymbol{enb}_s(tr)\\ (2)\ \forall s, p(s) \Rightarrow \Phi(s)\\ (3)\ \forall tr' \neq tr, \{\Phi\}\ tr'\ \{\Phi\}\\ (4)\ \forall s, \Phi(s) \Rightarrow q(s)\\ (5)\ \forall s, \Phi(s) \wedge \boldsymbol{enb}_s(tr) \Rightarrow r(s)\end{array}}{\forall \sigma, \forall i,\ \sigma \models_i p \mathbin{\dot{\Rightarrow}} q \; \mathcal{W}_{\overrightarrow{I_{tr}}} r}$$

$$\frac{\begin{array}{l}(1)\ \forall s, p(s) \Rightarrow \Phi(s) \vee q(s)\\ (2)\ \forall s, \Phi(s) \Rightarrow \boldsymbol{enb}_s(tr)\\ (3)\ \forall s, \forall tr' \neq tr, \Phi(s) \wedge \boldsymbol{enb}_s(tr')\\ \quad \Rightarrow \min(I_{tr'}) > 0 \wedge (tr', tr) \notin \mathcal{R}\\ (4)\ \forall tr' \neq tr, \{\Phi\}\ tr'\ \{\Phi \vee q\}\\ (5)\ \{\Phi\}\ tr\ \{q\} \wedge \max(I_{tr}) < \infty\end{array}}{\forall \sigma, \forall i,\ \sigma \models_i p \mathbin{\dot{\Rightarrow}} \Diamond_{\overleftarrow{I_{tr}}} q}$$

6 Conclusions

We have presented a mathematical model for specifying and reasoning over real-time systems. This model enables us to specify some very subtle timed-semantic

[4] Carlos D.Luna has defined in Coq the TCTL tree logic : `http://coq.inria.fr/pylons/contribs/view/CTLTCTL/v8.4`

differences of fine-grained constructs (see Section 1.1) allowed by modern real-time component-based languages such as FIACRE. Also, we provide logical and functional definitions to give semantic interpretations of real-time systems specified as TCTS*s* in terms of well known formalisms that allow to reason over their timing and temporal aspects. All definitions and theorems presented in this paper have been fully formalized and established in the COQ proof assistant [5].

We stress also in this conclusion that we do not address the whole FIACRE language [6] semantics in this paper. However, we consider the introduced model of TCTS (together with [12]) as the *semantic kernel* of FIACRE and the cornerstone of future research and developments. Transition systems with *real-time constraints and priorities* are very complicated and reasoning about them is tedious. But on the other hand such systems are widely used in real-life embedded systems, especially in spatial and avionics domains, which is a good motivator for developing a fully mechanized framework. This makes the presented work a very useful starting point for further theoretical or applied developments.

Future work will consist in certifying through the use of COQ some *existing patterns* used in the design of avionic embedded applications, such as the *periodic controller* presented in [8]. Also, an application of this work is to certify within COQ the translation of timed temporal formulæ into FIACRE observers [1]. Another envisioned application of our semantic framework concerns the verification of transformations between timed formalisms such as the one presented in [5], or model simplification, as for example, the flattening operator of component-based languages. At last, we are convinced that this work together with [12] could be a good basis for the certified compilation [19] of the FIACRE language.

References

1. Abid, N., Zilio, S.D., Botlan, D.L.: A Verified Approach for Checking Real-Time Specification Patterns. In: Proceedings of VeCos (2012)
2. Arnold, A.: Finite Transition Systems - Semantics of Communicating Systems. Prentice Hall international series in computer science. Prentice Hall (1994)
3. Basu, A., Bozga, M., Sifakis, J.: Modeling Heterogeneous Real-time Components in BIP. In: SEFM, pp. 3–12 (2006)
4. Bérard, B., Cassez, F., Haddad, S., Lime, D., Roux, O.H.: Comparison of Different Semantics for Time Petri Nets. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 293–307. Springer, Heidelberg (2005)
5. Bérard, B., Cassez, F., Haddad, S., Lime, D., Roux, O.H.: Comparison of the Expressiveness of Timed Automata and Time Petri Nets. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 211–225. Springer, Heidelberg (2005)
6. Berthomieu, B., Bodeveix, J.-P., Farail, P., Filali, M., Garavel, H., Gaufillet, P., Lang, F., Vernadat, F.: Fiacre: an Intermediate Language for Model Verification in the Topcased Environment. In: ERTS 2008 (2008)
7. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development (Coq'Art: The Calculus of Inductive Constructions). Texts in Theoretical Computer Science. Springer (2004)

[5] Available at `http://www.irit.fr/~Manuel.Garnacho/Coq/FORMATS13`.

8. Bodeveix, J.-P., Filali, M., Garnacho, M., Spadotti, R., Yang, Z.: On the Mechanization of an AADL Subset. Science of Computer Programming: special issue on Architecture Design Language (submitted, 2013)
9. Chaki, S., Clarke, E.M., Ouaknine, J., Sharygina, N., Sinha, N.: State/Event-Based Software Model Checking. In: Boiten, E.A., Derrick, J., Smith, G.P. (eds.) IFM 2004. LNCS, vol. 2999, pp. 128–147. Springer, Heidelberg (2004)
10. Emerson, E.A., Halpern, J.Y.: Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. In: STOC, pp. 169–180 (1982)
11. The FIACRE Specification Language for Real-Time Concurrent Systems, `http://projects.laas.fr/fiacre/`
12. Garnacho, M., Bodeveix, J.-P., Filali, M.: Mechanized Semantics of Concurrent Systems with Priorities. IRIT Research Report–2013-16–FR (2013), `http://www.irit.fr/~Manuel.Garnacho/Publications/MechPrio.pdf`
13. Geuvers, H., Koprowski, A., Synek, D., van der Weegen, E.: Automated Machine-Checked Hybrid System Safety Proofs. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 259–274. Springer, Heidelberg (2010)
14. Hale, R., Cardell-Oliver, R., Herbert, J.: An Embedding of Timed Transition Systems in HOL. FMSD 3(1/2), 151–174 (1993)
15. Henzinger, T.A., Manna, Z., Pnueli, A.: Temporal Proof Methodologies for Real-time Systems. In: POPL, pp. 353–366 (1991)
16. Henzinger, T.A., Manna, Z., Pnueli, A.: Timed Transition Systems. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 226–251. Springer, Heidelberg (1992)
17. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
18. The ISABELLE System, `http://isabelle.in.tum.de/`
19. Leroy, X.: Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In: 33rd Symposium Principles of Programming Languages, pp. 42–54. ACM Press (2006)
20. Paulin-Mohring, C.: Modelisation of Timed Automata in Coq. In: Kobayashi, N., Babu, C. S. (eds.) TACS 2001. LNCS, vol. 2215, pp. 298–315. Springer, Heidelberg (2001)
21. Pnueli, A.: The Temporal Logic of Programs. In: FOCS, pp. 46–57 (1977)
22. Rushby, J.: Mechanized Formal Methods: Progress and Prospects. In: Chandru, V., Vinay, V. (eds.) FSTTCS 1996. LNCS, vol. 1180, pp. 43–51. Springer, Heidelberg (1996)