

A Timed Extension for AltaRica

Franck Cassez, Claire Pagetti and Olivier Roux
IRCCyN
1 rue de la Noë
BP 92101
44321 Nantes Cedex 3
France
email: Name.Surname@irccyn.ec-nantes.fr

28th November 2002

Abstract

In this work we present a *timed* extension of the AltaRica formalism [1, 2]. Following the work of [2], we extend the semantics of AltaRica to *timed components* and *timed nodes*. Moreover we also extend the *priority mechanism* of AltaRica to the timed case. Finally we give a translation of a Timed AltaRica specification into a usual timed automaton that can then be analysed by existing tools for checking properties of timed systems.

1 Introduction

Context. The present technological progress leads the industrial systems to embed many software and hardware components. Due to the manifest complexity, the software is in charge of a huge number of tasks and modelling it often requires the use of particular techniques such as composition. Furthermore, a system has to guarantee some quality and safety conditions. Consequently on the need for specifying and verifying those systems, the so-called *formal methods* are thoroughly developed.

The first step towards the design of such systems is *specification*. The AltaRica language [1, 3, 2] is a high-level specification formalism that allows one to specify *constraint automata* [3] with the following features:

- a *component* has its own variables (internal or external), plus some others it can only read (*flow variables*) that are shared by the others;
- components can be defined hierarchically and composed together by a general synchronisation mechanism. One can express broadcast communication, give priority among some transitions, etc.

Moreover AltaRica has an unambiguous semantics [1, 2] defined in terms of (*interfaced*) transition systems. From this semantic model, it is possible to compile AltaRica to lower level formalisms for different verification purposes: fault-tree to perform reliability analysis [4], Petri nets, Markov graphs or finite state automata (that can be analysed by MEC [5, 6] for instance).

Nevertheless one cannot specify real-time constraints in AltaRica and of course this becomes crucial when some timing information contribute to the modelling and verification of the system. Moreover there is no real high-level specification language for timed and hybrid systems. This makes AltaRica a good candidate to fill this gap. Once the language has been extended with timing constraints, we can take advantage of the work carried out these last years about timed systems: it is now well-known how to deal with the verification of *timed automata* [7] and *hybrid automata* [8, 9] and many efficient tools are now available [10, 11, 12]. This adds a new feature to the AltaRica toolbox.

Our Contribution. Our work consists in extending the AltaRica formalism with *real-time constraints*. We thus extend the semantic model of AltaRica, the *interfaced transition system (ITS)*, into *timed interfaced transition system (TITS)* and give the semantics of Timed AltaRica in terms of TITS. We proceed by shifting all the results obtained for AltaRica (e.g. Interface Bisimulation homomorphism, rewriting of a node into a component, ...) to the timed case. Finally we present an algorithm to compile Timed AltaRica specifications into timed automata (UPPAAL [13]).

Outline of the paper. In the next section, we remind the basics about AltaRica and introduce a running example. Section 3 is the core of the paper and presents the timed extension of AltaRica. In sections 4 and 5 we respectively give (i) the algorithm for translating Timed AltaRica components into timed automata and (ii) an example of the use of priorities for timed specification. We conclude by some perspectives in section 6.

The proofs of the theorems are given in the appendices (pages 25 -).

2 An Overview of the AltaRica Language

In this section we give an example of an AltaRica specification. We take as a running example the train-gate-controller [14]. In this example, the aim is to keep the gate closed when a train is in a critical section.

2.1 AltaRica Specification

A specification in AltaRica is a *node*. A node is basically composed of:

- the *variables definitions* (type, range, ...), and *events definitions*,
- the *transition relation*,
- the *initial constraints* and *global constraints*.

In the example of Fig. 1, we define a component *train* to model the behaviour of a train in two equivalent manners in order to ease the understanding: an AltaRica description (see Fig. 1(a)) and a standard automaton (see Fig. 1(b)) (if we abstract away \mathbb{N}). A train is either *Far* of the critical section, or *Before* or *On* meaning it is respectively near or inside the critical zone. In the AltaRica specification, the variable `etat` ranging in $[0, 2]$ represents the locations *Far*, *Before*, *On* of the train. The events of the component TRAIN are *approach*, *in* and *exit*. We also use a variable `n` to denote that the state of the train is in $\{Before, On\}$. Initially the component is in configuration `etat=0,n=0`. Also the flow variable \mathbb{N} is in the *interface* of the node `train`. This means that other

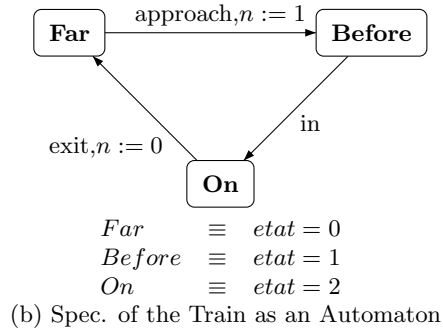
```

node TRAIN
flow N [0,1];
event approach, in, exit;
state etat [0,2], n [0,1];
trans etat=0 |- approach ->
      etat := 1, n := 1;
      etat=1 |- in ->
      etat := 2;
      etat=2 |- exit ->
      etat := 0, n := 0;

init etat=0, n=0;
assert N=n;
edon

```

(a) Spec. of the Train in AltaRica



(b) Spec. of the Train as an Automaton

Figure 1: Specification of a Train

nodes can read it (but not modify it). On the contrary the variables `state` and `n` are local variables of the train node and are not visible by other nodes. The value of the flow variable `N` is constrained to be equal to `n` at anytime (see the `assert` line).

We can also specify the nodes `GATE` and `CONTROLLER` for the *Gate* and the *Controller*: the (timed) nodes are given in Fig. 6 and 7. We only depict here the interfaces of those nodes (see Fig. 2) to complete the description of our train-gate-controller system. The node `Gate` has four events `Go_up`, `Go_down` (commands given by the controller to the gate to go up or down). For the node `Controller` the events are `approach`, `exit`, `Go_up` and `Go_down`. From those nodes we can build a synchronised system.

2.2 About the Hierarchy and the Composition in AltaRica

As emphasized in the introduction, one can describe a system by composing and building new nodes from sub-nodes. For example we can define a node `Main` (see Fig. 2(a)) specifying the train-gate-controller with two trains. It is composed of four sub-nodes (`t1`, `t2`, `g` and `c`, see Fig. 2(b)) which interact in two ways: flow coordination and synchronisation of events. The synchronisation constraint (after keyword `sync`) reads as follows:

- line 1 stands for three synchronisation vectors: those for which at least one event (the constraint ≥ 1) qualified with a “?” appears; the value “-” means the gate is idle (not involved in the transition);
- line 2 means that only the gate and the controller interact and the trains are not involved.

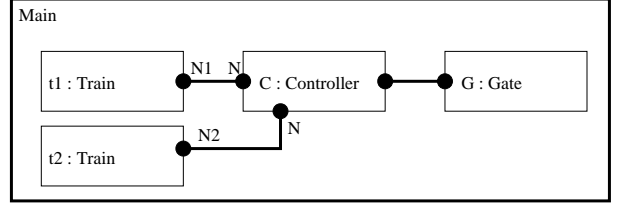
Finally the global assertion, line 3, constrains the flow variables so that `N` of the node `MAIN` is always equal to the number of trains on the critical section.

```

node MAIN
  sub t1,t2 : TRAIN, g : GATE,
    c : CONTROLLER;
  sync
  1: <t1.approach?,t2.approach?,-,
    c.approach> >= 1;
    <-,-,g.Go_down,c.Go_down>;
    <t1.exit?,t2.exit?,-,c.exit> >= 1;
  2: <-,-,g.Go_up,c.Go_up>;
  3: assert c.N=t1.N+t2.N;
edon

```

(a) AltaRica Description



(b) Hierarchical Overview

Figure 2: Node for the Train-Gate-Controller

2.3 Formal Semantics of AltaRica

The semantics of AltaRica specifications is given by Interfaced Transition Systems. For a detailed presentation of these notions the reader is referred to [1, 2].

2.3.1 Interfaced Transition Systems

Definition 1 (Interfaced transition system [2]). An interfaced Transition System (ITS) is a tuple $\mathcal{A} = \langle E, F, S, \pi, T \rangle$ with:

1. $E = E_+ \cup \{\epsilon\}$ is a finite set of events such that $\epsilon \notin E_+$;
2. F is a set of flow values;
3. S is the set of states;
4. $\pi : S \rightarrow 2^F$ associates to each state s in S all the admissible flow values in s . We assume $\forall s \in S, \pi(s) \neq \emptyset$.
5. $T \subseteq S \times F \times E \times S$ is the transition relation and satisfies:

- (a) $(s, f, e, s') \in T \Rightarrow f \in \pi(s)$
- (b) $\forall s \in S, f \in \pi(s), (s, f, \epsilon, s) \in T$

A configuration of an ITS is a pair $(s, f) \in S \times F$ such that $f \in \pi(s)$. \square

Remark 1. In AltaRica, if a transition (s, f, e, s') is firable then there exists a configuration (s', f') (the mapping π constrains the set S). This remark will carry over timed ITS. The set F may be considered as a set of properties (or observations) associated to the states by the mapping π . \square

2.3.2 Priorities

In AltaRica we can constrain the behaviours of a system by giving priorities to some transitions when more than one is possible. For instance, this concept is classical in scheduling [15]. Formally, a priority relation $<$ is a strict partial order over the events. A transition labelled e can be fired from a configuration (s, f) if it is *maximal*, i.e. no other transition e' such that $e < e'$ is firable in (s, f) .

Definition 2 (Priority relation [2]). A priority relation over E is a strict partial order over E such that $\forall v \in E_+, v \not\prec \epsilon$ and $\epsilon \not\prec v$ (with $E_+ = E \setminus \{\epsilon\}$). \square

Definition 3 (Priority Restriction Operator). Let $\mathcal{A} = \langle E, F, S, \pi, T \rangle$ be an ITS and $<$ a priority relation over E . We define the priority restriction operator \upharpoonright for the transition relation $T \subseteq S \times F \times E \times S$ and the priority relation $<$ by: $(s, f, e, s') \in T \upharpoonright \Leftrightarrow (s, f, e, s') \in T \wedge \forall s'' \in S, \forall e' \in E, (s, f, e', s'') \in T \Rightarrow e \not\prec e'$. We define $\mathcal{A} \upharpoonright \Leftarrow = \langle E, F, S, \pi, T \upharpoonright \Leftarrow \rangle$. \square

2.3.3 Formulas and Expressions

We consider hereafter the *expressions* $\mathbb{E}(X)$ built over the variables in a set X . These expressions can be either integer terms, boolean terms etc. The only thing we assume is that the variables in X take their values in a set \mathcal{D} . A *valuation* ν of a set of variables X is a mapping $\nu : X \rightarrow \mathcal{D}$ and the set of valuations of X is denoted \mathcal{D}^X . The *value* of an expression $e \in \mathbb{E}(X)$ in the context $\nu : X \rightarrow \mathcal{D}$ is denoted $e(\nu)$. Given a set $\mathbb{E}(X)$ we can define the set $\mathbb{F}(X)$ of *first order formulas* over $\mathbb{E}(X)$ using some suitable predicates (e.g. $\leq, =$ in the case of integer expressions) and the existential and universal quantifiers. For $f \in \mathbb{F}(X)$ we denote $free(f)$ the set of free variables in f . We assume that tt and ff which are predicates of arity 0 belong to $\mathbb{F}(X)$. In the sequel we often omit the base set X when we use $\mathbb{F}(X)$ as only the free variables used in a formula $f \in \mathbb{F}(X)$ are relevant.

The interpretation $\llbracket f \rrbracket$ of a formula $f \in \mathbb{F}(X)$ with $free(f) \subseteq X'$ is a subset of $\mathcal{D}^{X'}$: $\llbracket f \rrbracket \subseteq \mathcal{D}^{X'}$. Also we have $\llbracket tt \rrbracket = \mathcal{D}^X$ and $\llbracket ff \rrbracket = \emptyset$.

An *assignment* for the variables in X is a mapping $a : X \rightarrow \mathbb{E}(X)$. Intuitively an assignment of the form $x := y + z + 2$ will be defined by $a(x) = y + z + 2$. Given a valuation $\nu : X \rightarrow \mathcal{D}$, we denote by $a(\nu)$ the valuation defined by $a(\nu)(x) = a(x)(\nu)$. We denote by Id the *identity* assignment such that $\forall x, Id(x) = x$.

Now we define an abstract syntax for the AltaRica components and nodes again taken from [2].

2.3.4 AltaRica Components

AltaRica components give an abstract syntax for the basic systems (no hierarchy) introduced in the previous section.

Definition 4 (Component). A component is a tuple $\mathcal{C} = \langle V_S, V_F, E, A, M, < \rangle$ with:

1. V_S, V_F are finite sets for respectively state variables, flow variables, with the property of being 2 by 2 disjoint. We denote $V_C = V_S \cup V_F$;
2. $E = E_+ \cup \{\epsilon\}$ is a finite set of events and as usual ϵ is the empty action;
3. $A \in \mathbb{F}$ is an assertion such that $free(A) \subseteq V_C$;
4. $M \subseteq \mathbb{F} \times E \times \mathbb{E}(V_C)^{V_S}$ is a macro-transition relation such that $\forall (g, e, a) \in M$:
 - (a) $g \in \mathbb{F}$ is a guard such that $free(g) \subseteq V_C$,
 - (b) $e \in E_+$ is the event of the transition,
 - (c) $a : V_S \rightarrow \mathbb{E}(V_C)$ is an assignment for the variables in V_S .

and $(tt, \epsilon, Id) \in M$;

5. $<$ is a priority relation. \square

Remark 2. In [2], another set of flow variables is defined: it corresponds to unobservable flow variables that can be used as intermediary variables. We omit them in this work as they do not increase the expressiveness of the language. Indeed they can be defined as existentially quantified flow variables in the assertion of a node.

Now we can define the semantics of a component to be an ITS. For the semantic definitions, we assume that all variables V_S, V_F have a common domain \mathcal{D} .

Definition 5 (Semantics of Components). Let $\mathcal{C} = \langle V_S, V_F, E, A, M, < \rangle$ be a component. The semantics of \mathcal{C} is the interfaced transition system $\llbracket \mathcal{C} \rrbracket = \langle E, F, S, \pi, T \rangle$ constructed in the following way:

1. $F = \mathcal{D}^{V_F}$;
2. $S = \{s \in \mathcal{D}^{V_S} \mid \exists f \in \mathcal{D}^{V_F}, (s, f) \in \llbracket A \rrbracket\}$;
3. $\pi : S \rightarrow 2^F$ such that $\pi(s) = \{f \mid (s, f) \in \llbracket A \rrbracket\}$;
4. $T \subseteq S \times F \times E \times S$ such that $T = \llbracket M \rrbracket \upharpoonright <$ with:
 - (a) $\llbracket M \rrbracket = \cup_{t \in M} \llbracket t \rrbracket$;
 - (b) $\llbracket (g, e, a) \rrbracket = \{(s, f, e, s') \mid (s, f) \in \llbracket A \wedge g \rrbracket, s' = a(s, f)\}$

2.3.5 AltaRica nodes

A *node* is built from n components. The purpose of nodes is to give a semantics to hierarchical definitions and synchronisation in AltaRica.

Definition 6 (AltaRica Node). A node is a tuple $\mathcal{N} = \langle V_F, E, <, \mathcal{N}_0, \dots, \mathcal{N}_n, V \rangle$ with:

1. V_F is a set of flow variables,
2. $E = E_+ \cup \{\epsilon\}$ is a finite set of events,
3. $<$ is a priority relation over E ,
4. for all $i \in [1, n]$, \mathcal{N}_i is a component or a node; V_{F_i} is the set of flow variables of \mathcal{N}_i and E_i the set of events. We assume $\forall i \neq j \in [1, n], V_{F_i} \cap V_{F_j} = \emptyset$,
5. \mathcal{N}_0 is a special component called the control component. The set of events of this node is $E_0 = E$ and the priority relation of \mathcal{N}_0 is the empty relation. The set of flow variables of \mathcal{N}_0 is $V_{F_0} = V_F \cup V_{F_1} \cup V_{F_2} \cup \dots \cup V_{F_n}$,
6. $V = V_d \cup V_{imp}$ is the set of specified synchronisation vectors:
 - $V_d \subseteq E_0^? \times \dots \times E_n^? \times 2^{[0, n+1]}$ where $E_i^? = E_i \cup \{?e, e \in E_{i+}\}$; we define E_d by: $e \in E_d$ if $\exists \langle \dots, x_i, \dots \rangle \in V_d$ with $x_i \in \{e_i, ?e_i\}$;
 - $V_{imp} \subseteq E_0 \times \dots \times E_n \times \{0\}$ is the set of implicit synchronisation vectors with:
 - $\langle \epsilon, \dots, \epsilon, 0 \rangle \in V_{imp}$,
 - $\forall i \in [0, n], \forall e_i \in E_i \setminus E_d, \langle \epsilon, \dots, e_i, \dots, \epsilon, 0 \rangle \in V_{imp}$.

V_{imp} contains all the synchronisation vectors with non synchronised events. □

The semantics of an AltaRica node is obtained by building the synchronised product of the sub-nodes and deleting some transitions according to the priority relation given by the partial order of the node. The semantics of a node \mathcal{N} is an ITS $\llbracket \mathcal{N} \rrbracket$ (see [2]). The synchronisation set V generates a set of synchronisation vectors of $E_0 \times E_1 \times \dots \times E_n$ together with a priority relation on them¹. As already mentioned in the example of 1, a vector of the form $\langle \text{t1.exit?}, \text{t2.exit?}, -, \text{c.exit} \rangle \geq 1$ generates all the synchronisation vectors containing at least one event the name of which is qualified by a “?”. The priority relation for those vectors correspond to giving priority to the one with the maximal number of qualified events: in the previous case $\langle \text{t1.exit}, -, -, \text{c.exit} \rangle < \langle \text{t1.exit}, \text{t2.exit}, -, \text{c.exit} \rangle$ and $\langle -, \text{t2.exit}, -, \text{c.exit} \rangle < \langle \text{t1.exit}, \text{t2.exit}, -, \text{c.exit} \rangle$. The set V_{imp} consists of all the events that are not involved in any synchronisation: they must occur on their own, hence the synchronisation vectors of the form $\langle \epsilon, \dots, e, \dots, \epsilon, 0 \rangle$.

For a formal definition of how to generate the synchronisation vectors corresponding to V the reader is referred to [2]. We only need here to consider the set of synchronisation vectors and the priority relation generated by V .

In the definition of the *timed nodes* (section 3.6) we will focus on timed features and will consider that V has been “unfolded” into the set of synchronisation vectors it generates and the priority relation, i.e. we will use $\tilde{V} \subseteq E_0 \times E_1 \times \dots \times E_n$ and $\langle_{\tilde{V}}$ instead of V .

There is a fundamental result about nodes: they can be rewritten (syntactically) into components that preserve their semantics [2]. If \mathcal{N} is an AltaRica node, $\mathcal{C}_{\mathcal{N}}$ its rewriting into a component, then $\llbracket \mathcal{N} \rrbracket = \llbracket \mathcal{C}_{\mathcal{N}} \rrbracket$.

In the next section we focus on extending ITS and AltaRica components and nodes with time. We define our timed extension on these objects. Also we show that the results obtained in the untimed case [2, 1] still hold (e.g. a Timed AltaRica node can be rewritten into Timed AltaRica component).

3 Timed Extension of AltaRica

Our aim is to build a timed *extension* of AltaRica, which means we need to keep the framework defined for the untimed case: ITS, priorities and components. First we extend ITS into *Timed ITS* (see def. 7) and define *timed* priorities. Then we add timing specification to components (i.e. *clock variables*) and give the semantics of timed components into TITS. Finally we define *timed nodes* give their semantics and prove that they can be syntactically rewritten into an equivalent (timed bisimilar) component.

3.1 Preliminaries about Timed Systems

Before defining Timed AltaRica we recall some basics about timed systems [16]. More precisely we use the framework of timed automata [7] and the associated usual notations. The real-valued variables will be *clocks*: a *clock* is a positive real valued variable, and it evolves at a constant rate w.r.t. physical time.

Clock valuations and assignments. A *clock valuation* for the clocks in a set X is a mapping $v : X \mapsto \mathbb{R}_{\geq 0}$ that assigns a positive real value to each clock in X . Assignments for clocks will take a particular form². A *clock assignment* is a mapping $a : X \rightarrow \mathbb{N} \cup X$. We denote by $\mathcal{A}(X)$ the set

¹how this is done is defined in [2].

²they could be more general, but as in section 4 we want to translate Timed AltaRica specifications into Timed Automata, we restrict the assignments to a subset such that reachability is decidable for timed automata [7, 17].

of clock assignments. As defined in subsection 2.3.3, for a clock valuation v and an assignment a , we denote $a(v)$ the clock assignment $a(v)(x) = a(x)(v)$. For $t \in \mathbb{R}_{\geq 0}$ the clock valuation $v + t$ is defined by $\forall x \in X, (v + t)(x) = v(x) + t$.

The set of *clock constraints* $\mathcal{B}(X)$ over a set X of clocks is defined inductively by:

$$g := x \smile r \mid x - y \smile r \mid g \wedge g \mid g \vee g$$

with $x, y \in X, \smile \in \{<, <, >, \geq, =\}$, $r \in \mathbb{Q}$. Also we denote by $\mathcal{B}_C(X)$ the subset of $\mathcal{B}(X)$ that defines *convex clock constraints*. A clock constraint g is a particular formula and evaluates either to tt or ff : $\llbracket g \rrbracket \subseteq \mathbb{R}_{\geq 0}^X$ and $g(v) = tt \iff v \in \llbracket g \rrbracket$.

Timed Transition Systems and Timed Automata. A *timed transition system* [16] (TTS) is a tuple $(Q, E, q_0, \rightarrow, O)$, where Q is set of locations, E is the set of actions, q_0 is the initial location, $\rightarrow \subseteq Q \times (E \cup \mathbb{R}_{\geq 0}) \times Q$ and $O : Q \rightarrow 2^P$ is the *labelling function* that associates some properties to each state (some suitable temporal logic can then be defined on TTS). A *Timed automaton* [7] is a tuple (L, l_0, E, X, I, T, O) such that L is a (finite) set of *locations*, l_0 is the *initial location*, E is a finite set of *actions*, X is a finite set of *clocks*, $T \subseteq L \times (\mathcal{B}(X) \times E \times \mathcal{A}(X)) \times L$ is the *transition relation*, $I : L \rightarrow \mathcal{B}_C(X)$ is the *invariant constraint* and $O : L \rightarrow 2^P$ a *labelling function*.

The semantics of a timed automaton (L, l_0, E, X, I, T, O) is given by a TTS $(L \times \mathbb{R}_{\geq 0}, E, (l_0, v_0), \rightarrow, O')$ where $\forall x \in X, v_0(x) = 0$ and $\forall (l, v) \in L \times \mathbb{R}_{\geq 0}, O'(l, v) = O(l)$, \rightarrow is defined by: i) *discrete steps* of the form $(l, v) \xrightarrow{e} (l', v')$ iff $\exists (l, g, e, a, l') \in T$, such that $g(v) = tt, v' = a(v), v' \in \llbracket I(l') \rrbracket$, ii) *continuous steps* of the form $(l, v) \xrightarrow{\delta} (l, v'), \delta \in \mathbb{R}_{\geq 0}$ iff $\forall \delta' \leq \delta, v + \delta' \in \llbracket I(l) \rrbracket$.

A very useful result about timed automata (actually *updatable* timed automata [17]) is that reachability is decidable [7, 17] for this class of timed systems. Hence automatic verification tools have been designed to analyse timed automata, and among them UPPAAL [12] and KRONOS [18]. We will give in the last section a translation from a timed AltaRica specification into a timed automaton. This will allow us to use UPPAAL [12] or KRONOS [18] to check timed properties on the designed systems.

In the sequel, we define *Timed Interfaced Transition Systems* (TITS) that are extended TTS. The timed extension of AltaRica components are *timed components* that are the counter parts of timed automata: the semantics of timed components is given by TITS.

3.2 Timed Interfaced Transition Systems

Timed Interfaced Transition Systems are an extension of ITS with real-valued variables and flows.

Definition 7 (Timed Interfaced Transition System). A timed interfaced transition system (TITS) of continuous dimension (n, m) and time domain³ \mathbb{T} is a tuple $\mathcal{A} = \langle E_t, F_t, S_t, \pi, T \rangle$ with:

1. $E_t = E_+ \cup \{\epsilon\} \cup \mathbb{T}$ where E_+ is a finite set of events such that $\epsilon \notin E_+ \cup \mathbb{T}$ and $E_+ \cap \mathbb{T} = \emptyset$;
2. $F_t = F \times \mathbb{R}^m$ is the set of flow values, where F is the set of discrete flow values and \mathbb{R}^m is the set of continuous flow values;
3. $S_t = S \times \mathbb{R}^n$ is the set of states where S is the set of discrete states and \mathbb{R}^n is the set of continuous states;

³we assume $0 \in \mathbb{T}$ and $\mathbb{T} = \mathbb{N}$ or $\mathbb{Q}_{\geq 0}$ or $\mathbb{R}_{\geq 0}$ or $\{0\}$.

4. $\pi : S_t \rightarrow 2^{F_t}$ associates to each state $q \in S_t$ all the admissible flow values in q . We assume $\forall q \in S_t, \pi(q) \neq \emptyset$.

5. $T \subseteq S_t \times F_t \times E_t \times S_t$ is the transition relation and satisfies:

- (a) $(q, g, e, q') \in T \Rightarrow g \in \pi(q)$
- (b) $\forall q \in S_t, \forall g \in \pi(q), (q, g, \epsilon, q) \in T$
- (c) $\forall q \in S_t, \forall g \in \pi(q), (q, g, 0, q) \in T$

A configuration of a TITS is a pair $((s, \nu), (f, \mu)) \in S_t \times F_t$ such that $(f, \mu) \in \pi(s, \nu)$ □

Remark 3. If $n = 0$ and $m = 0$ and $\mathbb{T} = \{0\}$ we find back the definition of ITS. If $m = 0$ and we add an initial state to the TITS then we obtain the definition of TTS: F is to be interpreted as the set of atomic properties.

It is possible to consider an integer time domain, $\mathbb{T} = \mathbb{N}$. Notice that in this case even if we allow only integer time steps in the TITS, the values of the clocks can be in $\mathbb{R}_{\geq 0}$.

For a dense time domain $\mathbb{T} = \mathbb{Q}_{\geq 0}$ is suitable. For a continuous time domain one can take $\mathbb{T} = \mathbb{R}_{\geq 0}$.

3.3 Timed Priorities

In the discrete version of AltaRica, priorities among events play an important role [2]: they allow the easy modelling of priorities among concurrently enabled transitions. It is natural in a *timed* setting to try and introduce *timed priorities* i.e. priorities among transitions involving some timing information. Again we want to extend the existing AltaRica specification language and add timed priorities.

Timed priorities in timed systems have been introduced for timed automata [19] and a comprehensive study of timed priorities can be found in [20, 21, 22]. The most common timed priority is *urgency* [23]: basically, it says that if a transition is enabled in a timed automaton, time can not elapse and this transition must be fired immediately. Without loss of expressiveness we define *urgent events*: if an event in an Timed AltaRica specification is *urgent* then all the transitions labelled by this event are urgent. We then extend definition 2 to allow priority between time labels (in \mathbb{T}) and discrete events:

Definition 8 (Simple Timed Priority Relation). A simple timed priority relation $<$ over E is a strict partial order over $E \cup \{time\}$ such that $<$ is a priority relation over E and a strict partial order over E_+^{time} with $E_+^{time} = E_+ \cup \{time\}$ and $\forall v \in E_+, v \not< time$. □

Then $a > time$ means that event a is *urgent* and has to be fired immediately when enabled (a semantic definition of priorities will appear in section 3.3). Also note that if $a > time$ and $b > a$ then $b > time$: the urgency of event a entails urgency of greater events.

This allows us to model what is called *eagerness* in [23]. In these papers other notions of priorities are defined: (i) a *delayable* transition is one that can be fired when its guard is true and before a certain *deadline*; (ii) a *lazy* transition is one that has no deadline (it may or may not be fired). We will add in definition 11 (*time*) *guards* into Timed AltaRica components which will enable us to define *lazy* transitions. It is proved in [21] that a delayable transition can be encoded into a lazy and an eager transition. As we already know (def. 8) how to define eager transitions, we are able to express the three types of priorities proposed in [23].

More elaborate ways of prioritising transitions are given in [20]. The aim is to express priority between events when several are enabled by using timing information. It is an extension of the priority relation notion: we want to express that $e < e'$ only if e' will not be enabled in some future. Intuitively, we will write $e <_5 e'$ for: the transition labelled e' if enabled within 5 time units has priority over e . We now extend the notion of simple timed priority relation:

Definition 9 (Timed Priority Relation). A timed priority relation $<$ over E is a 3-ary relation in $E_+^{time} \times (\mathbb{N} \cup \{\infty\}) \times E_+^{time}$ satisfying the following conditions (we denote $a_1 <_k a_2$ for $(a_1, k, a_2) \in <$):

- the binary relation $<_0$ is a simple timed priority relation,
- $a <_k b \wedge (a = \text{time} \vee b = \text{time}) \implies k = 0$,
- $\forall k \in \mathbb{N}, <_k$ is a strict partial order,
- $a_1 <_k a_2 \implies (\forall k' < k, a_1 <_{k'} a_2)$, □

Remark 4. Notice that in [20], another condition can be imposed on $<$ (a transitivity condition). This condition is related to the building of live timed systems and is not relevant in our setting. Also we do not want to build live timed systems but only to provide a restriction operator (by giving priorities). In the framework of [20], the aim is to build live timed systems which entails modifying the deadlines of the transitions: this also adds new behaviours to the composed systems.

Note also we restrict the bounds to \mathbb{N} which in theory is enough for specifying timed systems.

The static priority of AltaRica coincides with the particular timed priority where all the delays are equal to zero, *i.e.* $k = 0$.

As in section 2.3.2, we define the timed priority restriction operator.

Definition 10 (Timed Priority Restriction Operator). Let $\mathcal{A} = \langle E_t, F_t, S_t, \pi, T \rangle$ be a TITS of continuous dimension (m, n) , time domain \mathbb{T} and $<$ a timed priority relation over E . We define the timed priority restriction operator \upharpoonright for the transition relation $T \subseteq S_t \times F_t \times E_t \times S_t$ and the timed priority relation $<$ by:

$$(q, g, e, q') \in T \upharpoonright < \Leftrightarrow (q, g, e, q') \in T \wedge \begin{cases} \text{if } e = t \in \mathbb{T}, \forall t' \in \mathbb{T}, t' \leq t, \text{ if } (q, g, t', q') \in T \\ \quad \text{then } \forall e' \in E_+, (q', g', e', q'') \in T \implies \text{time} \not\prec_0 e'. \\ \text{otherwise if } (q, g, t, q'') \in T, t \in \mathbb{T}, t \leq k, \\ \quad \text{then } \forall e', (q'', g'', e', q''') \in T \implies e \not\prec_k e'. \end{cases}$$

We denote $\mathcal{A} \upharpoonright < = \langle E_t, F_t, S_t, \pi, T \upharpoonright < \rangle$. □

Remark 5. Again, if $\mathbb{T} = \{0\}$, we find the definition of the priority relation restriction (def. 3).

We can lift the following theorem for ITS stated in [2] to TITS:

Theorem 1 (Timed Priority and Timed Bisimulation). Let $\mathcal{A}_1 = \langle E_t, F_t, S_{1t}, \pi_1, T_1 \rangle$ and $\mathcal{A}_2 = \langle E_t, F_t, S_{2t}, \pi_2, T_2 \rangle$ be two TITS and $<$ a timed priority relation over E . If $h : \mathcal{A}_1 \longrightarrow \mathcal{A}_2$ is a timed bisimulation homomorphism then $h : \mathcal{A}_1 \upharpoonright < \longrightarrow \mathcal{A}_2 \upharpoonright <$ is also a timed bisimulation homomorphism.

The proof is given in appendix A.2.

3.4 Timed Components

Timed AltaRica components are the timed extensions of AltaRica components (see def. 4). Our extension consists in adding *clocks* to AltaRica components. Hence our model is closely related to the timed automaton model. Adding real-valued variables instead of clocks is quite straightforward: the resulting model is then close to the hybrid automaton model. In this paper we focus on the timed extension and the addition of clocks. We consider the formulas \mathbb{F} , expressions \mathbb{E} and set of values \mathcal{D} settled in the section 2.3.3.

Definition 11 (Timed Component). A timed component is a tuple $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \prec \rangle$ with:

1. V_S, V_F are finite sets for respectively state variables, flow variables with the property of being disjoint. We denote $V_T = V_S \cup V_F$; C_S, C_F are finite sets for respectively clock variables, clock flow variables with the property of being disjoint. We denote $C_T = C_S \cup C_F$; also we assume $V_T \cap C_T = \emptyset$;
2. $E = E_+ \cup \{\epsilon\}$ where E_+ is a finite set of events and as usual ϵ is the empty action;
3. $A = A_{V_T} \cap A_{C_T} \in \mathbb{F}$ is an assertion such that $\text{free}(A) \subseteq V_T \cup C_T$; $A_{V_T} \in \mathbb{F}$, $\text{free}(A_{V_T}) \subseteq V_T$. A_{C_T} is a set of constraints of the form $P \implies I$ where $P \in \mathbb{F}$, $\text{free}(P) \subseteq V_T$ and $I \in \mathbb{F}$, $\text{free}(I) \subseteq C_T$ and I defines a convex region of \mathbb{R}^n if $|C_S| = n$;
4. $M \subseteq (\mathbb{F} \times \mathcal{B}(C_T)) \times E \times (\mathbb{E}(V_T)^{V_T} \times \mathcal{A}(C_T))$ is a macro-transition relation such that $\forall ((g, \gamma), e, (a, R)) \in M$:
 - (a) (g, γ) is a guard such that $g \in \mathbb{F}$ and $\text{free}(g) \subseteq V_C$; $\gamma \in \mathcal{B}(C_T)$;
 - (b) $e \in E$ is the event of the transition;
 - (c) $a : V_S \rightarrow \mathbb{E}(V_C)$ is an assignment for the variables in V_S . $R \in \mathcal{A}(C_T)$ is the clock assignment of the transition;
5. \prec is a timed priority relation. □

Notice that the semantics of A is a subset of $(\mathcal{D}^{V_S} \times \mathbb{R}^n) \times (\mathcal{D}^{V_F} \times \mathbb{R}^m)$ as well as the semantics of a guard (g, γ) .

Definition 12 (Semantics of Timed Components). Let $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \prec \rangle$ be a timed component. Let $|C_S| = n$ and $|C_F| = m$. The semantics of \mathcal{T} over the time domain \mathbb{T} is the timed interfaced transition system $\llbracket \mathcal{T} \rrbracket = \langle E_t, F_t, S_t, \pi, T \rangle$ of dimension (n, m) constructed in the following way:

1. $F_t = \mathcal{D}^{V_F} \times \mathbb{R}^m$;
2. $S_t = \{(s, \nu) \in \mathcal{D}^{V_S} \times \mathbb{R}^n \mid \exists (f, \mu) \in \mathcal{D}^{V_F} \times \mathbb{R}^m, ((s, \nu), (f, \mu)) \in \llbracket A \rrbracket\}$;
3. $\pi : S_t \rightarrow 2^{F_t}$ such that $\pi(q) = \{(f, \mu) \mid (q, (f, \mu)) \in \llbracket A \rrbracket\}$;
4. $T \subseteq S_t \times F_t \times E_t \times S_t$ such that $T = \llbracket M \rrbracket \upharpoonright \prec$ with:
 - (a) $\llbracket M \rrbracket = \cup_{t \in M} \llbracket t \rrbracket \cup \cup_{\delta \in \mathbb{T}} \llbracket \delta \rrbracket$;
 - (b) $\llbracket ((g, \gamma), e, (a, R)) \rrbracket = \{((s, \nu), (f, \mu), e, (s', \nu')) \mid ((s, \nu), (f, \mu)) \in \llbracket A \wedge g \wedge \gamma \rrbracket, s' = a(s, f) \wedge \nu' = R(\nu, \mu)\}$, (with $R(\nu, \mu)$ the new clock assignment after resetting the variables in R).

$$(c) \delta \in \mathbb{T}, \llbracket \delta \rrbracket = \{((s, \nu), (f, \mu), \delta, (s', \nu')) \mid ((s, \nu), (f, \mu)) \in \llbracket A \rrbracket, s' = s \wedge \nu' = \nu + \delta \wedge \forall \delta' \leq \delta, \pi(s, \nu + \delta') \neq \emptyset\}. \quad \square$$

Remark 6. In the case $V_F = \emptyset$ we obtain the definition of timed automata (again if we add an initial state); the semantics of such a timed component is then a TTS (again V_F is to be interpreted as some properties or observations on each state.) Also notice that the value of the flow variables could change on a time step of duration δ . This is already the case for the definition of the components where a flow variable can change on an ϵ transition. This is up to the designer to ensure that the assertion maps a state to a unique flow value. Anyway in a forward collecting semantics (computing symbolically the reachable states) this should cause no problem as if (s, f) is reachable with f containing at least two different values, letting time pass or doing an ϵ transition will lead to the same reachable set.

As for the untimed case we have the following lemma:

Lemma 1. Let $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, < \rangle$ be a timed component and $<$ is a timed priority relation. Then $\llbracket \langle V_S \cup C_S, V_F \cup C_F, E, A, M, < \rangle \rrbracket = \llbracket \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset \rangle \rrbracket \upharpoonright <$.

The proof is straightforward from definitions 10 and 12.

3.5 Syntactical Timed Priority

In [20] the authors show that it is possible to encode a priority relation by strengthening the guards of a component: this way one can syntactically encode the priority relation.

We tackle this problem in the timed case. Let $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset \rangle$ be a timed component and $<$ a timed priority relation. We first assume that $<$ contains no urgent events i.e. $\forall e \in E, \text{time} \not\prec e$. Our aim is to compute the transition relation $M \upharpoonright <$ syntactically i.e. by finding new guards that define $M \upharpoonright <$. We first rewrite our timed component so that we are sure that when a guard evaluates to true, the corresponding transition can indeed be fired, i.e. the resulting new state satisfies assertion A . This is done by adding weakest precondition into the existing guards. Then we show how to encode timed priority again by strengthening the guards. Finally we give some hints on how to handle urgency.

3.5.1 Weakest Precondition

The key point is to know if a transition $((g, \gamma), e, (a, R))$ can really be fired, and the fact that the guard evaluates to true is not sufficient: a new state can be reached from $((s, \nu), (f, \mu))$, only if after the assignments given by (a, R) there are some flow values such that $\pi(a(s, f), R(\nu, \mu)) \neq \emptyset$. This latter condition depends on assertion A of the timed component and can be seen as a *weakest precondition*.

First assume we have an untimed component (def. 4). Let $t = (g, e, a)$ be a transition of this component, and A the assertion. For $Q \subseteq S \times F$, we define $Pre_t(Q) = \{(s, f), \exists f', (a(s, f), f') \in Q\}$. Assume $Pre_t(\llbracket A \rrbracket)$ can be defined by a formula $\phi_t \in \mathbb{F}$, and $free(\phi_t) \subseteq V_T$. Now if we take $t' = (g \wedge \phi_t, e, a)$, we are sure that when $g \wedge \phi_t$ evaluates to true the transition t can be fired as $(s, f) \in \llbracket g \wedge \phi_t \rrbracket \implies \pi(a(s, f)) \neq \emptyset$.

We can extend this to the timed component. For $t = ((g, \gamma), e, (a, R))$ we define $Pre_t(Q) = \{((q', \nu'), (f', \mu')), \exists f'', \mu'', ((a(s', f'), R(\nu', \mu')), (f'', \mu'')) \in Q\}$. Assume $Pre_t(\llbracket A \rrbracket)$ can be written as $\phi_t \wedge \theta_t$ where ϕ_t is such that $free(\phi_t) \subseteq V_T$ and $free(\theta_t) \subseteq C_T$.

Then if we define $t' = ((g \wedge \phi_t, \gamma \wedge \theta_t), e, (a, R))$, we can ensure that if the guard of t' evaluates to true, t can be fired.

Now we show how to encode $Pre_t(\llbracket A \rrbracket)$ into guards of the form as (g, γ) with $g \in \mathbb{F}$, $free(g) \subseteq V_T$ and $\gamma \in \mathcal{B}(C_T)$. Assume $A = p_1 \wedge p_2 \wedge \dots \wedge p_n \wedge (q_1 \implies i_1) \wedge \dots \wedge (q_l \implies i_l)$. Thus $Pre_t(\llbracket A \rrbracket)$ is a conjunction of the form $p'_1 \wedge p'_2 \wedge \dots \wedge p'_k \wedge (q'_1 \implies i'_1) \wedge \dots \wedge (q'_m \implies i'_m)$. Let $P' = p'_1 \wedge p'_2 \wedge \dots \wedge p'_k$. We can rewrite $Pre_t(\llbracket A \rrbracket)$ as:

$$\bigvee_{\substack{J \cup I = [1..m] \\ I \cap J = \emptyset}} \underbrace{P' \wedge \bigwedge_{j \in J} \neg q'_j \wedge \bigwedge_{r \in I} q'_r \wedge \bigwedge_{r \in I} i'_r}_{G_{I,J}} \underbrace{\phantom{P' \wedge \bigwedge_{j \in J} \neg q'_j \wedge \bigwedge_{r \in I} q'_r \wedge \bigwedge_{r \in I} i'_r}}_{\Gamma_{I,J}}$$

This is a formula of the form $\bigwedge_{p=1..s} g_p \wedge \gamma_p$ with $g_p \in \mathbb{F}$, $free(g_p) \subseteq V_T$ and $\gamma_p \in \mathcal{B}(C_T)$. Now we create s transitions from $t = ((g, \gamma), e, (a, R))$ defined by:

$$\forall p \in [1..s], t_p = ((g \wedge g_p, \gamma \wedge \gamma_p), e, (a, R))$$

and t can be fired if and only if one of the t_p can be fired (leading to the same values for the state variables.)

3.5.2 Encoding Timed Priority

Definition 13 (Modal Operator [21, 22]). Let $X = \{x_1, x_2, \dots, x_n\}$. Let $\nu \in \mathbb{R}^n$ and $k \in \mathbb{N}$. Let $\phi \in \mathcal{B}(X)$ and the time domain \mathbb{T} . We define the (state) predicate $\diamond_k \phi$ by:

$$(\diamond_k \phi)(\nu) \iff \exists t \in \mathbb{T}, t \leq k, \phi(\nu + t)$$

□

Now take a transition $t = ((g, \gamma), e, (a, R)) \in M$. Assume $e <_k e'$ and $t' = ((g', \gamma'), e', (a', R')) \in M$. Then t can be fired only if e' is not enabled within k time units. This can be written as $\neg \diamond_k \gamma'$. The encoding of $e <_k e'$ can now be written by strengthening the timed guard of t . It becomes $\gamma \wedge \neg \diamond_k \gamma'$. More generally for a timed priority relation $<$ we rewrite the guard γ of t as $\gamma_{<}$ with:

$$\gamma_{<} = \gamma \wedge \bigwedge_{\substack{((g', \gamma'), e', (a', R')) \in M \\ e <_k e'}} \neg \diamond_k \gamma'$$

Remark 7. As stated in remark 4, page 10, we do not modify the invariants of the system.

3.5.3 Syntactical Timed Priority

As stated in [20], the formula $\diamond_k \gamma$ can be written as simple formula in $\mathcal{B}(V_T)$. If we denote by $M \upharpoonright_{<}$ the transition relation obtained by

1. strengthening the guards by the weakest precondition for *fireability* as defined in section 3.5.1,
2. strengthening the guards to encode the timed priority relation as defined in 3.5.2,

we obtain a new timed component $\mathcal{T} \upharpoonright_{<} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M \upharpoonright_{<}, \emptyset \rangle$ such that:

Lemma 2 (Syntactical Priority). Let $<$ be a timed priority relation. Then $\llbracket \mathcal{T} \upharpoonright_{<} \rrbracket = \llbracket \mathcal{T} \rrbracket \upharpoonright_{<}$. □

From lemma 1, we obtain the following corollary:

Corollary 1. $\llbracket \langle V_S \cup C_S, V_F \cup C_F, E, A, M, < \rangle \rrbracket = \llbracket \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset \rangle \upharpoonright_{<} \rrbracket$ □

3.5.4 Encoding Urgency

Urgency (preventing time elapsing when a discrete transition is enabled) can be handled in a similar manner as fireability. For the sake of simplicity we assume here that g defines an integer value (e.g. a location in a timed automaton). We only give here a sketch of the encoding: assume $t = ((g, \gamma), e, (a, R)) \in M$ and e is urgent:

1. $U_t = \llbracket \gamma \rrbracket \cap \llbracket A \rrbracket$ defines the values where t must be fired immediately when we are in g , without letting time pass,
2. we add a fresh clock $x \notin C_T$,
3. for any transition t' leading to g we create a transition t'_u leading to g_u and almost equal to t' except that the guard of t'_u is the conjunction of the guard of t' and the weakest precondition leading to U_t and on t'_u we reset the new fresh clock x ;
4. also we strengthen the guard of t by the weakest precondition leading outside U_t ; then when firing transition t to reach g we end up in a non-urgent configuration;
5. we add the assertion $g_u \implies x = 0$; when in g_u time cannot elapse but U_t is satisfied and at least one transition can be fired;
6. it remains to strengthen the assertion of g with $\neg\gamma$ (or more precisely the negation of the rising edge of γ , see [22] for the details); then we can let time pass as long as γ evaluates to false and as soon as it is true t will be fired.

In the sequel we will show how to translate a time component into a timed automaton with the aim of verifying it. As many verification tools provide *urgent* transitions, we will not need to implement the previous algorithm, but only to tag the urgent transitions with the urgent feature.

3.6 Timed Nodes

Timed nodes are straightforward extensions of nodes. Indeed, if we assume as stated in def. 6 of a node, that the synchronisation constraint is expanded, the new constraint added by the time transitions is trivial: the synchronised time transitions for n nodes are of the form $(\delta, \delta, \dots, \delta)$, $\delta \in \mathbb{T}$ where \mathbb{T} is the time domain and they do not need to be specified.

Definition 14 (AltaRica Timed Node). *A timed node is a tuple $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \dots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}}) \rangle$ with:*

1. V_F is a set of flow variables,
2. C_F is the set of clock flow variables,
3. $E = E_+ \cup \{\epsilon\}$ is a finite set of events,
4. $<$ is a timed priority relation over E ,
5. for all $i \in [1, n]$, \mathcal{N}_i is a timed component or a timed node; V_{F_i} (resp. C_{F_i}) is the set of flow (resp. clock flow) variables of \mathcal{N}_i and E_i the set of events. We assume $\forall i \neq j \in [1, n], V_{F_i} \cap V_{F_j} = C_{F_i} \cap C_{F_j} = \emptyset$,

6. \mathcal{N}_0 is a special timed component called the control component. The set of events of this node is $E_0 = E$ and the priority relation of \mathcal{N}_0 is the empty relation. The set of flow variables of \mathcal{N}_0 is $V_{F_0} = V_F \cup V_{F_1} \cup V_{F_2} \cup \dots \cup V_{F_n}$, and the set of clock flow variables is $C_{F_0} = C_F \cup C_{F_1} \cup C_{F_2} \cup \dots \cup C_{F_n}$.

7. $\tilde{V} \subseteq E_0 \times E_1 \times \dots \times E_n$ is an expanded synchronisation set together with a priority relation $\prec_{\tilde{V}}$. (see def. 2). \square

Remark 8. Notice that $\prec_{\tilde{V}}$ is a priority relation and not a timed priority relation. This is because $(\tilde{V}, \prec_{\tilde{V}})$ expresses the discrete synchronisation constraint. The timing constraint \prec can be a timed priority relation.

Syntactically there is not much changes between timed and untimed nodes. The differences appear in the semantics where the timed transitions are synchronised:

Definition 15 (Semantics of an AltaRica Timed Node). Let $\mathcal{N} = \langle V_F, C_F, E, \prec, \mathcal{N}_0, \dots, \mathcal{N}_n, (\tilde{V}, \prec_{\tilde{V}}) \rangle$ be a timed node and $\llbracket \mathcal{N}_i \rrbracket = \langle E_{i_t}, F_{i_t}, S_{i_t}, \pi_i, T_i \rangle$ of dimension (n_i, m_i) for $i \in [0, n]$. The semantics of \mathcal{N} is the timed interfaced transition system $\llbracket \mathcal{N} \rrbracket = \langle E_t, F_t, S_t, \pi, T \rangle$ of dimension $(\sum_{k=0}^n n_k, |C_F|)$ defined by:

1. $F_t = \mathcal{D}^{V_F} \times \mathbb{R}^m$, with $m = |C_F|$,
2. for $q_i \in S_{i_t}$, $\pi(q_0, q_1, \dots, q_n) = \{(f, \mu) \in \mathcal{D}^{V_F} \times \mathbb{R}^m, \forall i \in [1, n], \exists \eta_i \in \pi_i(q_i), ((f, \mu), \eta_1, \eta_2, \dots, \eta_n) \in \pi_0(q_0)\}$,
3. $S_t = \{q \in S_{0_t} \times S_{1_t} \times \dots \times S_{n_t}, \pi(q) \neq \emptyset\}$;
4. $T \subseteq S_t \times F_t \times E_t \times S_t$ is defined by:
 - (a) let \prec_0 be the timed priority relation defined by: $(e_0, e_1, \dots, e_n) \prec_0 (e'_0, e'_1, \dots, e'_n) \iff e_0 \prec e'_0$,
 - (b) let $T_{\mathcal{N}}$ be the set of transitions defined by
 - discrete step: let $e = (e_0, e_1, \dots, e_n)$

$$\langle (s_0, s_1, \dots, s_n), f, e, (s'_0, s'_1, \dots, s'_n) \rangle \in T_{\mathcal{N}} \iff \begin{cases} \exists f_0 = (f, f_1, \dots, f_n) \in \pi_0(s_0) \\ \forall i \in [0, n], (s_i, f_i, e_i, s'_i) \in T_i \wedge \\ f_i \in \pi_i(s_i) \end{cases}$$

- time step: $\delta \in \mathbb{T}$

$$\langle (s_0, s_1, \dots, s_n), f, \delta, (s'_0, s'_1, \dots, s'_n) \rangle \in T_{\mathcal{N}} \iff \begin{cases} \exists f_0 = (f, f_1, \dots, f_n) \in \pi_0(s_0) \\ \forall i \in [0, n], (s_i, f_i, \delta, s'_i) \in T_i \wedge \\ f_i \in \pi_i(s_i) \end{cases}$$

(c) then $T = (T_{\mathcal{N}} \upharpoonright \prec_{\tilde{V}}) \upharpoonright \prec_0$. \square

Moreover the synchronised product of the nodes is compositional with respect to timed bisimulation:

Theorem 2. Let $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \dots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}}) \rangle$ and $\mathcal{N}' = \langle V_F, C_F, E, <, \mathcal{N}'_0, \dots, \mathcal{N}'_n, (\tilde{V}, <_{\tilde{V}}) \rangle$ be timed nodes such that $\forall i \in [0..n]$ there is a timed homomorphism h_i from $\llbracket \mathcal{N}_i \rrbracket$ to $\llbracket \mathcal{N}'_i \rrbracket$. Then there exists a timed homomorphism h from $\llbracket \mathcal{N} \rrbracket$ to $\llbracket \mathcal{N}' \rrbracket$. \square

The proof is given in appendix A.3.

From the following definitions, we can shift the rewriting theorem of [2] to timed nodes:

Theorem 3. Let \mathcal{N} be a timed node. Then \mathcal{N} can be rewritten into a timed component $\mathcal{C}_{\mathcal{N}}$ such that $\llbracket \mathcal{N} \rrbracket = \llbracket \mathcal{C}_{\mathcal{N}} \rrbracket$. \square

The proof is given in appendix A.4.

4 From Timed Nodes to Timed Automata

In this section, we present translation from a timed AltaRica specification into a timed automaton [7]. This way we can extract a timed automaton from a timed AltaRica specification and carry out some verification of temporal properties using tools for analysing timed systems like UPPAAL [13], CMC [24] or KRONOS [18].

4.1 Building a Timed Automaton from a Timed Component

Thanks to theorem 3, we focus on the translation of a timed component into a timed automaton.

Let $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, < \rangle$ be a timed component. Let $|C_S| = n$ and $|C_F| = m$. According to lemma 2, we can assume that $<$ is the empty relation.

By definition 11, the assertion of a timed component consists in two parts:

- A_{V_T} which gives a constraint on the discrete variables,
- A_{C_T} which associates a *timed invariant* to a predicate on the discrete variables.

Thus we write $A_{C_T} = \bigwedge_{j=1}^p P_j \implies I_j$ with $free(P_j) \subseteq V_T$ and $free(I_j) \subseteq C_T$.

As we want to build a timed automaton (which is timed bisimilar to the original node) from a timed component, we need to define the locations of this timed automaton. They are built from the assertion on the discrete variables, and must be labelled with a timed invariant. We first give our translation and later account for its features.

We define the predicates (with free variables in V_T) r_T^F with $T \subseteq [1..p]$, $F \subseteq [1..p]$, $T \cap F = \emptyset$, $T \cup F = [1..p]$ (T and F form a partition of the set on indices of A_{C_T}) by: $r_T^F = (\bigwedge_{j \in T} P_j) \wedge (\bigwedge_{j \in F} \neg P_j)$. The set of locations of the timed automaton we want to build is given by $l_T^F = A_{V_T} \wedge r_T^F$ for all the T, F that partition the set $[1..p]$ (thus it is exponential in the number of predicates of A_{C_T}). Notice that this definition gives a partition of the set of states defined by A_{V_T} .

Now for each location l_T^F and each transition $t = ((g, \gamma), e, (a, R))$ in M , we compute the weakest precondition, defined in section 3.5.1, $Pre_t(l_T^F)$ ⁴ which gives the set of states that lead to l_T^F when firing t . $Pre_t(l_T^F)$ constrains only the variables in V_T .

We can now give the definition of the timed automaton $\mathcal{A}_{\mathcal{T}}$ associated with the timed component \mathcal{T} :

- the set L of locations of $\mathcal{A}_{\mathcal{T}}$ is given by the set of l_T^F for all T, F that partition the set $[1..p]$,

⁴ $\forall Q \subseteq S \times F$, $Pre_t(Q) = \{(s, f), \exists f', (a(s, f), f') \in Q\}$

- the set of clocks of \mathcal{A}_T is $C_T \cup C_F$,
- the set of actions is E ,
- the invariant I is given by: if $T \neq \emptyset$, $I(l_T^F) = \bigwedge_{k \in T} I_k$, otherwise $I(l_\emptyset^{[1..p]}) = tt$,
- the transition relation T is defined by: $\forall l_T^F, l_{T'}^F \in L$,

$$t = ((g, \gamma), e, (a, R)) \in M \iff (l_T^F, (g \wedge \text{Pre}_t(l_{T'}^F) \wedge \gamma, e, (a, R)), l_{T'}^F) \in T$$

- the labelling mapping $O : L \rightarrow 2^{V_F}$ is given by: $O(l_T^F) = \exists (V_S \cup C_S).(l_T^F)$.

Notice that we add constraints on the discrete variables in the guards, which is not formally allowed by the definition of timed automata we have given in section 3.1, but this definition can trivially be extended to include this. Also we allow assignments of the discrete variables on a transition, which implies extending definition 3.1. Finally if we add an initial location we obtain a complete definition for our timed automaton.

Theorem 4. *Let $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \leq \rangle$ be a timed component and $\mathcal{A}_T = (L, l_0, E, X, I, T, O)$ its translation in timed automaton. Then $\llbracket \mathcal{T} \rrbracket$ and $\llbracket \mathcal{A}_T \rrbracket$ are timed bisimilar.*

The proof is given in appendix A.5.

4.2 The Train Example

We now apply the previous definitions to the train example of Fig.3. Notice that we write in bold font the time features of the Train timed component.

```

node TRAIN
  flow  N [0,1];
  event approach, in, exit;
  state etat [0,2], n [0,1], t clock;
  trans t >= 70 and etat=0 |- approach ->
        etat := 1, t := 0, n := 1;
        20 <= t <= 30 and etat=1 |- in ->
        etat := 2, t := 0;
        10 <= t <= 20 and etat=2 |- exit ->
        etat := 0, t := 0, n := 0;
  init  etat=0, t=0;
  assert N=n;
        etat=1 => t <= 30;
        etat=2 => t <= 20;
edon

```

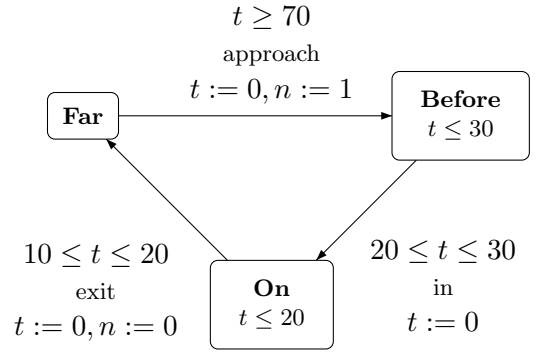


Figure 4: Train timed automaton

Figure 3: Timed AltaRica train specification

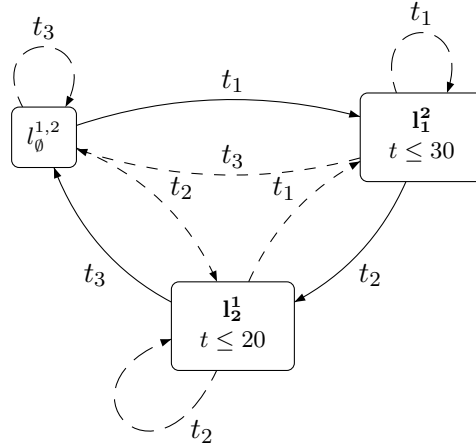
Given this timed description, we first compute the locations of the associated timed automaton from the assertion. The assertion consists in a boolean condition and two invariants: $A_{V_T} \equiv N = n$ and $A_{C_T} \equiv (P_1 \Rightarrow t \leq 30) \wedge (P_2 \Rightarrow t \leq 20)$, where $P_1 \equiv etat = 1$ and $P_2 \equiv etat = 2$. We enumerate

all the possible partitions in two subsets of the set $[1, 2]$:

$$\begin{aligned}
T = \emptyset, F = [1, 2] & \quad r_{\emptyset}^{1,2} \equiv \neg(etat = 1) \wedge \neg(etat = 2) \equiv \{etat = 0\} \\
T = \{1\}, F = \{2\} & \quad r_1^2 \equiv etat = 1 \wedge \neg(etat = 2) \equiv \{etat = 1\} \\
T = \{2\}, F = \{1\} & \quad r_2^1 \equiv etat = 2 \wedge \neg(etat = 1) \equiv \{etat = 2\} \\
T = [1, 2], F = \emptyset & \quad r_{1,2}^{\emptyset} \equiv etat = 1 \wedge etat = 2 \equiv \emptyset
\end{aligned}$$

Finally, we obtain three locations with invariants : $(l_{\emptyset}^{1,2} = \{etat = 0, N = n\}, I(l_{\emptyset}^{1,2}) = true)$, $(l_1^2 = \{etat = 1, N = n\}, I(l_1^2) = t \leq 30)$ and $(l_2^1 = \{etat = 2, N = n\}, I(l_2^1) = t \leq 20)$.

The next step consists in computing the graph structure. When a discrete transition is fired, the system must verify the assertion after applying the assignment. We use the weakest precondition operator Pre , defined by $\forall Q \subseteq S \times F$, $Pre_t(Q) = \{(s, f), \exists f', (a(s, f), f') \in Q\}$, in order to determine the transitions reaching each location. For instance, let us consider transition $t_1 = (g_1, e_1, a_1) = (etat = 0 \wedge t \geq 70, approach, a_1)$. Then, $Pre_{t_1}(l_{\emptyset}^{1,2}) = \{(etat, n, t, N) | \exists N', (etat' = 1, n' = 1, t' = 0, N') \in l_{\emptyset}^{1,2}\} = \emptyset$. Thus we cannot reach location $l_{\emptyset}^{1,2}$ by firing transition t_1 . We carry on the computation: $Pre_{t_1}(l_1^2) = \{l_{\emptyset}^{1,2}, l_1^2, l_2^1\}$ and $Pre_{t_1}(l_2^1) = \emptyset$. We iterate the process on each transition and we obtain the timed automaton of Fig. 5 which can be simplified into the automaton of Fig. 4.



$$\begin{aligned}
t_1 & \quad t \geq 70 \wedge etat = 0, approach, etat := 1, t := 0, n := 1 \\
t_2 & \quad 20 \leq t \leq 30 \wedge etat = 1, in, etat := 2, t := 0 \\
t_3 & \quad 10 \leq t \leq 20 \wedge etat = 2, exit, etat := 0, t := 0, n := 0
\end{aligned}$$

Figure 5: Abrupt Translation

Notice that the timed automaton of the gate specification contains only three locations even if there are 4 states in the specification: this is because we have chosen not to interpret the variables. Then in location l_{\emptyset} variable $etat$ can take value 0 or 2 and all this results in a 4 state automaton as expected.

4.3 Advantages of our Translation

The choices we have made can be accounted for by the following reasons:

- we do not want to have an expensive computation to produce the translation; our translation scheme is easy to implement and does not require extensive computation;

- also, we do not want to deal with clocks in the translation as it is the purpose of the tools for analysing timed systems to do some computation on continuous time domains;
- we do not want to constrain the discrete variables to be in a finite domain before giving the translation: indeed this could be the case that the variables are in a finite domain only because of the timing constraints. Thus we do not want to compute the domain of the variables in our translation. This is why the locations are predicates on the discrete variables and transitions constrain updates of this variables. Notice also that this could be the case that the timed automaton associated with a timed component has a finite bisimilar quotient whereas the untimed component has no finite bisimilar quotient (e.g. if a transition contains an update of the form $x := x + 1$). With our translation, we do not need to assume that the untimed component admits a finite bisimilar quotient.

5 Case Study : Timed Priorities Effects

In this section we give an example of the use of time priorities in Timed AltaRica and the modelling power they give. We consider again the train-gate-controller introduced in section 2:

- there are two tracks crossing at the gate,
- the trains can come from any side on these two tracks,
- the aim is to ensure the gate is closed when at least one train is on the near section. Also we do not want to open the gate if a train is crossing and another is going to cross in a near future (this is where the priorities will be used).

Let $k \in \mathbb{N}$ be a parameter, we fix some timed priorities among the two events *approach* and *Go_up* within a delay k . First, we translate this system in timed automata by applying the translation developed in the previous section 4. Second, we analyse the system using UPPAAL [13].

5.1 Translation of the Train-gate-controller into Timed Automata

We can either specify the controller in a separate timed component or see it as the particular sub-node \mathcal{N}_0 of the main timed node. It is easy to check that both designs are equivalent. In the following, we will use the second choice.

We specify below the timed component *Gate* in Fig. 6 and the timed node *Main* in Fig. 7. Furthermore, we add the timed priorities that $Go_up <_k approach$ (cf. Def 9).

We translate the timed component *Gate* and the sub-node \mathcal{N}_0 of the timed node *Main*. The whole system is then the synchronised product of the four automata: two trains, as computed in the previous section, a gate and a controller, constrained by the timed priority. The priority restriction consists in modifying some guards on this product, as explained in Def. 10. Rather than unfolding the synchronised automata, we compute symbolic guards on the controller automaton. So that, the synchronisation of the new automaton, the trains and the gate is semantically equivalent to the timed priority system.

5.2 Symbolic computation

The timed priority, $Go_up <_k approach$, applies to the states from which both transitions *Go_up* and *approach* are fireable. The controller is the only component that has these two events and we

```

node GATE
event Go_down, Go_up, down, up;
state etat [0,3], y clock;
trans etat=0 |- Go_up -> ;
      etat=0 |- Go_down ->
      etat := 1, y := 0;
      etat=1 |- Go_down -> ;
      y <= 10 & etat=1 |- down ->
      etat := 2;
      etat=1 |- Go_up -> etat :=3, y := 0;
      etat=2 |- Go_down -> ;
      etat=2 |- Go_up -> etat := 3, y :=0;
      etat=3 |- Go_up -> ;
      etat=3 |- Go_down ->
      etat := 1, y := 0;
      y <= 10 & etat=3 |- up -> etat := 0;
init etat=0, y = 0;
assert etat =1 => y <= 10;
      etat =3 => y <= 10;
edon

```

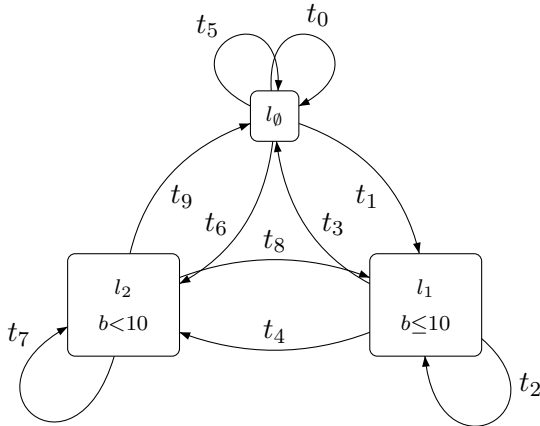
Figure 6: Gate specification

```

node MAIN
flow N integer;
event approach, exit, Go_up, Go_down;
priorities Go_up (<,k) approach;
state etat [0,2], z clock;
trans etat=0 |- approach -> etat := 1, z := 0;
      etat=0 & N>1 |- exit -> ;
      etat=0 & N=1 |- exit -> etat := 2, z := 0;
      etat = 1 |- approach -> ;
      etat = 1 |- exit -> ;
      z <= 10 & etat=1 |- Go_down -> etat := 0;
      z <= 10 & etat=2 |- Go_up -> etat := 0;
      etat=2 |- approach -> etat := 1, z := 0;
sub t1, t2 : TRAIN, g : GATE;
sync
<approach,t1.approach ?,t2.approach ?,-> >= 2;
<Go_down,-,-,g.Go_down>;
<exit,t1.exit ?,t2.exit ?,-> > 1;
<Go_up,-,-,g.Go_up>;
init etat = 0, z = 0;
assert N=t1.N+t2.N;
      etat =1 => z <10;
      etat =2 => z <= 10;
edon

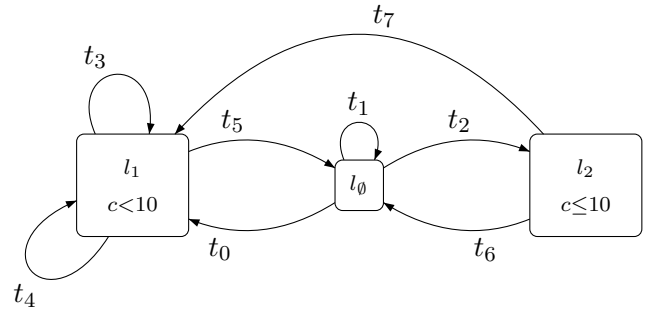
```

Figure 7: Controller specification



t_0 ($etat = 0, Go_up, id$)
 t_1 ($etat = 0, Go_down, etat := 1, y := 0$)
 t_2 ($etat = 1, Go_down, id$)
 t_3 ($y <= 10 \wedge etat = 1, down, etat := 2$)
 t_4 ($etat = 1, Go_up, etat := 3, y := 0$)
 t_5 ($etat = 2, Go_down, id$)
 t_6 ($etat = 2, Go_up, etat := 3, y := 0$)
 t_7 ($etat = 3, Go_up, id$)
 t_8 ($etat = 3, Go_down, etat := 1, y := 0$)
 t_9 ($y <= 10 \wedge etat = 3, up, etat := 0$)

Figure 8: Gate Translation into Timed Automaton



t_0 ($etat = 0, approach, etat := 1, z := 0$)
 t_1 ($etat = 0 \wedge N > 1, exit, id$)
 t_2 ($etat = 0 \wedge N = 1, exit, etat := 2, z := 0$)
 t_3 ($etat = 1, approach, id$)
 t_4 ($etat = 1, exit, id$)
 t_5 ($z <= 10 \wedge etat = 1, Go_down, etat := 0$)
 t_6 ($z <= 10 \wedge etat = 2, Go_up, etat := 0$)
 t_7 ($etat = 2, approach, etat := 1, z := 0$)

Figure 9: Controller Translation into Timed Automata

have $Pre_{approach}(l_\emptyset \cup l_1 \cup l_2) \cap Pre_{Go_up}(l_\emptyset \cup l_1 \cup l_2) = l_2$. So, the states of synchronised system affected by the timed priorities are of the form $\langle t1.-, t2.-, g.-, c.l_2 \rangle$ (we only need to compute the symbolic guards on the transitions that have this state as source state).

There are three possible transitions $\langle c.t_7, -, t_2.t_1, - \rangle$, $\langle c.t_7, t_1.t_6, -, - \rangle$ and $\langle c.t_6, -, -, g.Go_up \rangle$. We apply the syntactic transformations explained in section 4. For instance the guard with the label Go_up becomes $c \leq 10 \wedge \diamond_k(t_1.t \geq 70 \wedge t_2.t \geq 70) = c \leq 10 \wedge t_1.t < 70 - k \wedge t_2.t < 70 - k$. We modify the guards in the timed automaton of the controller accordingly and the result is the same timed automaton depicted on Fig. 9 in which the transition t_6 is modified in $t_6 = (z \leq 10 \wedge t_1.t < 70 - k \wedge t_2.t < 70 - k \wedge etat = 2, Go_up, etat := 0)$.

5.3 Influence of Timed Priorities

Timed priorities constraint the system. The first question that arises is what kind of behaviour can we loose.

In our example, the first issues are about the gate's role. In the system unless priority, $k = 0$, the gate opens infinitely often while after a certain value, there exists a path where the gate never opens. Using UPPAAL, we find out that for $k \leq 9$, the gate opens infinitely often while for $k \geq 10$ there is at least one trace where the gate stays closed forever. The screen-shots are exhibited Fig. 10 and Fig. 11

Knowing that there exists some paths where the gate stays closed forever, one can naturally wonder if the gate can always stay closed? It is the case when $k \geq 40$.

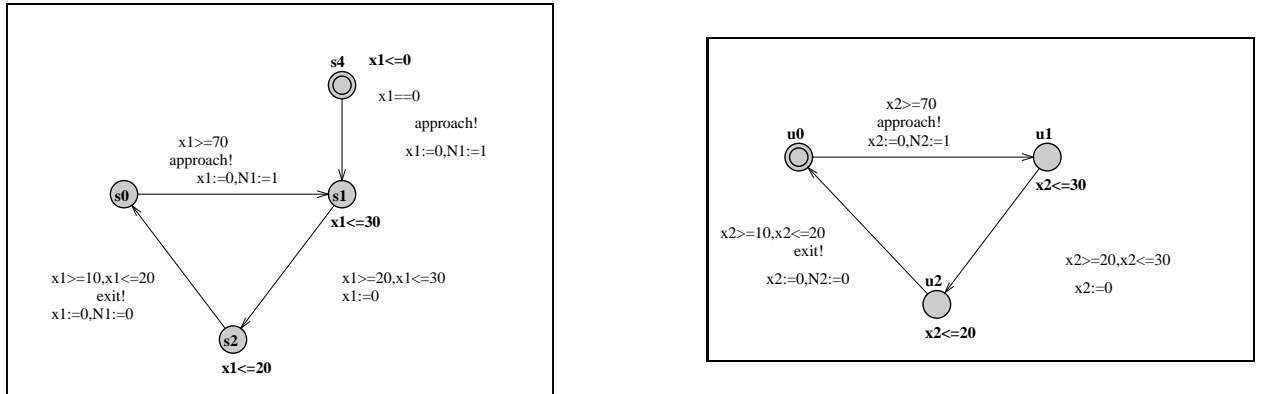


Figure 10: Train Automata in UPPAAL

The third and the most interesting problem concerns the liveness of the system. As stated in remark 4, page 10, some deadlock may occur when prioritising the system. The train-gate-controller less priority is deadlock free and loses this property as soon as $k > 0$. It comes from the invariant $z \leq 10$ in $c.l_2$. Indeed, following the syntactical construction of the deadline in [20], the invariant should become $etat = 2 \Rightarrow z \leq 10 \vee (z \leq 10 \wedge t_1.t < (70 - k) \wedge t_2.t < (70 - k))$. Then, we need to split the state $etat = 2$ into three states which respect the convex assumption: $etat = 2, 1 \Rightarrow z \leq 10 \wedge t_1.t < (70 - k) \wedge t_2.t < (70 - k)$, $etat = 2, 2 \Rightarrow z \leq 10 \wedge t_1.t \geq (70 - k)$ and $etat = 2, 3 \Rightarrow z \leq 10 \wedge t_2.t \geq (70 - k)$. Each transition ended in the $etat = 2$ or enabled from it are duplicated in or from the three states. (We delete the impossible transitions). We therefore obtain the system depicted in Fig 12. For this system, whatever is the value of k , there is no deadlock.

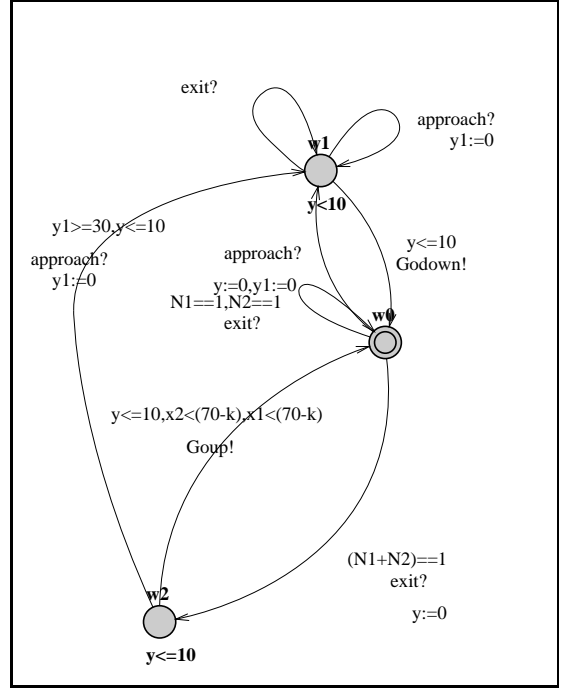
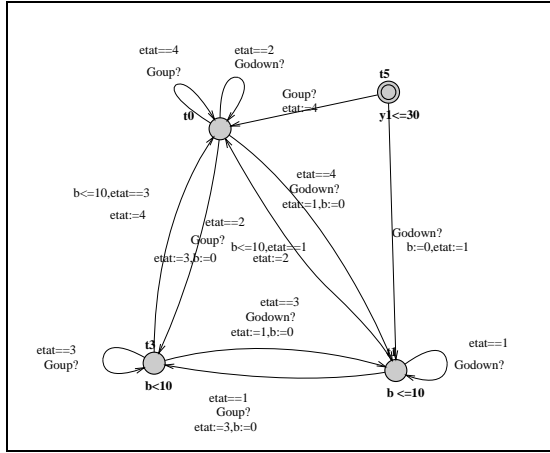


Figure 11: Gate and Controller Automata in UPPAAL

6 Conclusion

We have shown how to add clocks to AltaRica and build a timed extension of this formalism: Timed AltaRica. This timed extension has the same features as the untimed ones and we were able to prove all the results obtained for the untimed case:

- two timed bisimilar timed interfaced transition systems remain timed bisimilar when we apply a timed priority restriction (theorem 1),
- a timed component with timed priorities can syntactically be rewritten into a timed component without timed priorities and has the same semantics (lemma 2),
- the synchronised product (for nodes) is compositional with respect to timed bisimulation (theorem 2),
- a timed node can be rewritten into a timed component that has the same semantics (theorem 3).

Moreover we have defined a translation of timed components into usual timed automata (section 4) so that we can use tools for analysing timed automata (like UPPAAL) to carry out our verification.

Our future work is three-fold:

- complete the extension of AltaRica by adding features allowing the user to specify *hybrid systems* [8]; this should be a rather easy extension of our Timed AltaRica formalism,
- study the problem of preserving liveness when using priorities (following the framework of [20]),
- implement the translation we have given in section 4 in the AltaRica toolbox.

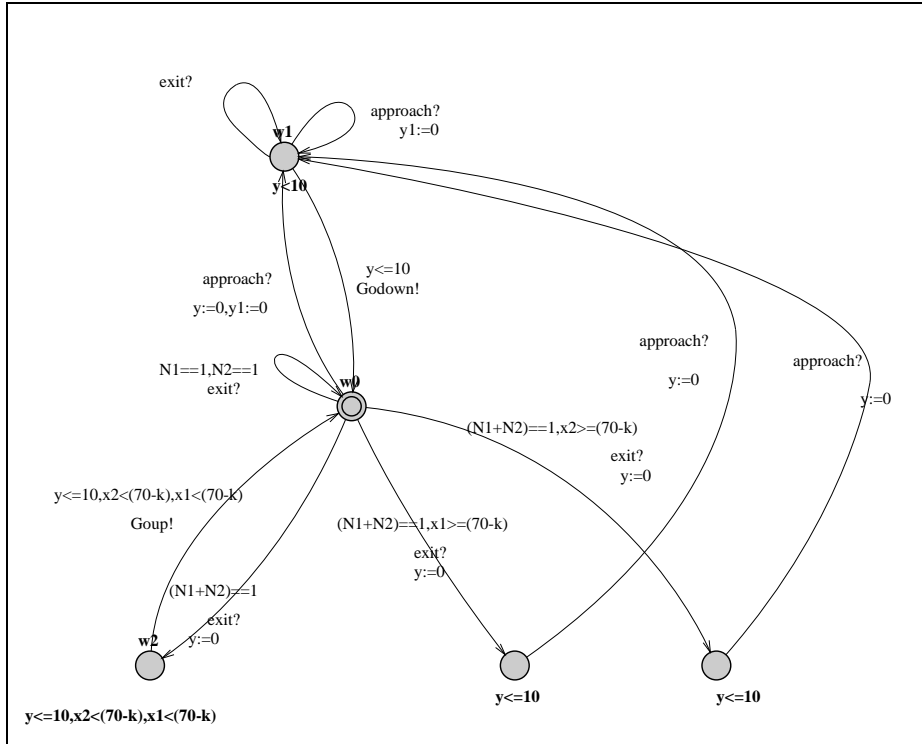


Figure 12: Deadlock Free Controller Automaton in UPPAAL

References

- [1] A. Arnold, A. Griffault, G. Point, and A. Rauzy. The altarica formalism for describing concurrent systems. *Fundamenta Informaticae*, 40:109–124, 2000. 1, 4, 7
- [2] G. Point. *Altarica : Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement*. PhD thesis, University of Bordeaux I, Janvier 2000. 1, 4, 5, 6, 7, 9, 10, 16
- [3] G. Point and A. Rauzy. Altarica - constraint automata as a description language. *European Journal on Automation*, 1999. Special issue on the *Modelling of Reactive Systems*. 1
- [4] A. Griffault, S. Lajeunesse, G. Point, A. Rauzy, J.-P. Signoret, and P. Thomas. The altarica language. In *Proceedings of the International Conference on Safety and Reliability, ESREL'98*. Balkema Publishers, June 20-24 1998. 1
- [5] A. Arnold, D. Begay, and P. Crubille. *Construction and analysis of transition systems with MEC*. World Scientific, 1994. 1
- [6] A. Arnold. *An experience with MEC in a real industrial project*. 1995. 1
- [7] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science B*, 126:183–235, 1994. 2, 7, 8, 16
- [8] T. A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press. 2, 22

- [9] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995. [2](#)
- [10] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2):110–122, 1997. [2](#)
- [11] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In A. J. Hu and M. Y. Vardi, editors, *Proc. 10th International Conference on Computer Aided Verification, Vancouver, Canada*, volume 1427, pages 546–550. Springer-Verlag, 1998. [2](#)
- [12] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, W. Yi, and C. Weise. New generation of uppaal, 1998. [2](#), [8](#)
- [13] Paul Pettersson and Kim G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000. [2](#), [16](#), [19](#)
- [14] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer-Verlag, 2001. [2](#)
- [15] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *13th Annual Symp. on Theoretical Aspects of Computer Science, STACS'96*, volume 1046 of *lncs*, pages 347–359, 1996. Invited paper. [4](#)
- [16] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16th IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press, December 1995. [7](#), [8](#)
- [17] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable ? In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855, pages 464–479. Springer, 2000. [7](#), [8](#)
- [18] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III: Verification and Control*, volume 1066, pages 208–219, Rutgers University, New Brunswick, NJ, USA, 22–25 October 1995. Springer. [8](#), [16](#)
- [19] Sebastien Bornot and Joseph Sifakis. On the composition of hybrid systems. In *HSCC*, pages 49–63, 1998. [9](#)
- [20] Sebastien Bornot, Gregor Gler, and Joseph Sifakis. On the construction of live timed systems. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 109–126, 2000. [9](#), [10](#), [12](#), [13](#), [21](#), [22](#)
- [21] Sébastien Bornot, Joseph Sifakis, and Stavros Tripakis. Modeling urgency in timed systems. *Lecture Notes in Computer Science*, 1536:103–129, 1998. [9](#), [13](#)
- [22] Sebastien Bornot and Joseph Sifakis. An algebraic framework for urgency. *Information and Computation*, 163(1):172–202, 2000. [9](#), [13](#), [14](#)

- [23] Joseph Sifakis and Sergio Yovine. Compositional specification of timed systems (extended abstract). In *Symposium on Theoretical Aspects of Computer Science*, pages 347–359, 1996. 9
- [24] F. Laroussinie and K. G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *Proc. IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pages 439–456. Kluwer Academic Publishers, 1998. 16

A Appendices

A.1 Timed Bisimulations

Definition 16 (Timed Interfaced Bisimulation). Let $\mathcal{A}_1 = \langle E_t, F_t, S_{1_t}, \pi_1, T_1 \rangle$ and $\mathcal{A}_2 = \langle E_t, F_t, S_{2_t}, \pi_2, T_2 \rangle$ be two TITS. A timed interfaced bisimulation relation for \mathcal{A}_1 and \mathcal{A}_2 is a relation $R \subseteq S_{1_t} \times S_{2_t}$ that satisfies:

1. $\forall q_1 \in S_{1_t}, \exists q_2 \in S_{2_t}, (q_1, q_2) \in R$ and $\forall q_2 \in S_{2_t}, \exists q_1 \in S_{1_t}, (q_1, q_2) \in R$
2. $\forall (q_1, q_2) \in R, \pi_1(q_1) = \pi_2(q_2)$
3. $\forall (q_1, g, e, q'_1) \in T_1, \forall q_2 \in S_{2_t}$ such that $(q_1, q_2) \in R$ then $\exists (q_2, g, e, q'_2) \in T_2, (q'_1, q'_2) \in R$
4. $\forall (q_2, g, e, q'_2) \in T_2, \forall q_1 \in S_{1_t}$ such that $(q_1, q_2) \in R$ then $\exists (q_1, g, e, q'_1) \in T_1, (q'_1, q'_2) \in R$. Two TITS are timed bisimilar iff there exists a timed interfaced bisimulation relation on their set of states.

Like in the untimed case, an interfaced bisimulation can be expressed as an homomorphism between two TITS.

Definition 17 (Timed Interfaced Bisimulation Homomorphism). Let $\mathcal{A}_1 = \langle E_t, F_t, S_{1_t}, \pi_1, T_1 \rangle$ and $\mathcal{A}_2 = \langle E_t, F_t, S_{2_t}, \pi_2, T_2 \rangle$ be two TITS. A timed interfaced bisimulation homomorphism $h : \mathcal{A}_1 \rightarrow \mathcal{A}_2$ is a mapping $h : S_{1_t} \rightarrow S_{2_t}$ such that:

1. h is surjective
2. $\forall q_1 \in S_{1_t}, \pi_1(q_1) = \pi_2(h(q_1))$
3. $\forall (q_1, g, e, q'_1) \in T_1, (h(q_1), g, e, h(q'_1)) \in T_2$
4. $\forall q_1 \in S_{1_t}, q'_2 \in S_{2_t}, (h(q_1), g, e, q'_2) \in T_2 \Rightarrow \exists q'_1 \in S_{1_t}, h(q'_1) = q'_2 \wedge (q_1, g, e, q'_1) \in T_1$

□

The following theorem follows immediately:

Theorem 5. Two TITS \mathcal{A}_1 and \mathcal{A}_2 are in timed interfaced bisimulation if and only if there exists a TITS \mathcal{B} and two timed interfaced bisimulation homomorphisms $h_1 : \mathcal{A}_1 \rightarrow \mathcal{B}$ and $h_2 : \mathcal{A}_2 \rightarrow \mathcal{B}$.

A.2 Proof of Theorem 1

Proof of theorem 1. Let $h : \mathcal{A}_1 \longrightarrow \mathcal{A}_2$ be a timed bisimulation homomorphism. We show that h is also a timed bisimulation homomorphism from $\mathcal{A}_1 \upharpoonright <$ onto $\mathcal{A}_2 \upharpoonright <$.

For points 1 and 2 of definition 17 just notice that $<$ only restricts the transition relation and does not involve the set of states and the mapping $\pi_{i=1,2}$.

Now for point 3, let $(q_1, g, e, q'_1) \in T_1 \upharpoonright <$, then $(h(q_1), g, e, h(q'_1)) \in T_2$. Assume that $(h(q_1), g, e, h(q'_1)) \notin T_2 \upharpoonright <$, then according to definition 10 there are two possibilities:

1. either $e = t \in \mathbb{T}$ and $\exists e' >_0$ time, $t' < t$, $(h(q_1), g, t', q''_2) \in T_2 \wedge (q''_2, g', e', q'''_2) \in T_2$. Since h is an homomorphism and $(h(q_1), g, t', q''_2)$ and (q''_2, g', e', q'''_2) are in T_2 , then $\exists q''_1, q'''_1 \in S_{1_t}, h(q''_1) = q''_2, h(q'''_1) = q'''_2 \wedge (q_1, g, t', q''_1)$ and (q''_1, g', e', q'''_1) are in T_1 . Hence (q_1, g, e, q'_1) cannot be in $T_1 \upharpoonright <$ which contradicts the first assumption.
2. otherwise $e \in E_+$ and $\exists e', e <_k e' \wedge (h(q_1), g, t, q''_2) \in T_2, t \in \mathbb{T}, t \leq k \wedge (q''_2, g', e', q'''_2) \in T_2$. Since h is an homomorphism and $(h(q_1), g, t, q''_2)$ and (q''_2, g', e', q'''_2) are in T_2 , there $\exists q''_1, q'''_1 \in S_{1_t}, h(q''_1) = q''_2, h(q'''_1) = q'''_2$ and (q_1, g, t, q''_1) and (q''_1, g', e', q'''_1) are in T_1 . This contradicts again the fact that $(q_1, g, e, q'_1) \in T_1 \upharpoonright <$. This ends the proof of point 3.

Now let $q_1 \in S_{1_t}, q'_2 \in S_{2_t}$ such that $(h(q_1), g, e, q'_2) \in T_2 \upharpoonright <$, then $\exists q'_1 \in S_{1_t}, h(q'_1) = q'_2 \wedge (q_1, g, e, q'_1) \in T_1$. Again assume that for all $q'_1, h(q'_1) = q'_2, (q_1, g, e, q'_1) \notin T_1 \upharpoonright <$:

1. if $e \in \mathbb{T}$, this means that $\exists t_1 \leq t, (q_1, g, t_1, q''_1) \in T_1$ and $\exists e' >_0$ time that is firable from (q''_1, g') . Then from $h, (h(q_1), g, t_1, h(q''_1)) \in T_2 \wedge (h(q''_1), g', e', q'''_2) \in T_2$ and it contradicts $(h(q_1), g, e, q'_2) \in T_2 \upharpoonright <$.
2. Otherwise e is in E_+ and there exists $e <_k e', t \leq k$ such that $(q_1, g, t, q''_1), (q''_1, g', e', q'''_1) \in T_1$. It follows that $(h(q_1), g, t, h(q''_1)), (h(q''_1), g', e', h(q'''_1)) \in T_2$. It contradicts again the hypothesis $(h(q_1), g, e, q'_2) \in T_2 \upharpoonright <$. This ends the proof of point 4 and of the theorem. □

A.3 Proof of theorem 2

Proof of theorem 2. Let $\forall i \in [0, n], \llbracket \mathcal{N}_i \rrbracket = \langle E_t, F_{i,t}, S_{i,t}, \pi_i, T_i \rangle$ and $\forall i \in [0, n], \llbracket \mathcal{N}'_i \rrbracket = \langle E_t, F_{i,t}, S'_{i,t}, \pi'_i, T'_i \rangle$ be the TITS defined by the semantics of the timed nodes. For the timed nodes \mathcal{N} and \mathcal{N}' , we use the lemma 1 which allows us to consider their semantics \mathcal{A} resp. \mathcal{A}' without timed priorities. Thus $\llbracket \mathcal{N} \rrbracket = (\langle E_t, F_t, S_t, \pi, T \rangle \upharpoonright <_{\bar{v}}) \upharpoonright <_0 = (\mathcal{A} \upharpoonright <_{\bar{v}}) \upharpoonright <_0$ and $\llbracket \mathcal{N}' \rrbracket = (\langle E_t, F_t, S'_t, \pi', T' \rangle \upharpoonright <_{\bar{v}}) \upharpoonright <_0 = (\mathcal{A}' \upharpoonright <_{\bar{v}}) \upharpoonright <_0$.

Next, the timed nodes \mathcal{N}_i and \mathcal{N}'_i are timed bisimilar. So $\forall i \in [0, n]$, let $h_i : \llbracket \mathcal{N}_i \rrbracket \longrightarrow \llbracket \mathcal{N}'_i \rrbracket$ be $n + 1$ timed bisimulation homomorphisms. We consider $h : S_t \longrightarrow S'_t$ defined by $h(\langle q_0, \dots, q_n \rangle) = \langle h_0(q_0), \dots, h_n(q_n) \rangle$. We show that h is a timed bisimulation homomorphism from \mathcal{A} onto \mathcal{A}' .

1. we first show that h is surjective. Let $q' = \langle q'_0, \dots, q'_n \rangle \in S'_t$, we know that each h_i are surjectives so that $\exists q_0, \dots, q_n \in S_{0,t} \times \dots \times S_{n,t}, q' = \langle h_0(q_0), \dots, h_n(q_n) \rangle = h(\langle q_0, \dots, q_n \rangle)$.

2. second, let $q \in S_t$, we show that $\pi(q) = \pi'(h(q))$.

$$\begin{aligned} \pi(q) &= \pi_0(q_0) \wedge \dots \wedge \pi_n(q_n) \\ &= \pi'_0(h_0(q_0)) \wedge \dots \wedge \pi'_n(h_n(q_n)) \\ &= \pi'(h(q)) \end{aligned}$$

3. Let $(q, g, e, q_1) \in T$, since there is no timed priority, we naturally have $(h(q), g, e, h(q_1)) \in T'$.
4. Let $q \in S_t, q'_1 \in S'_t$ with $(h(q), g, e, q'_1) \in T'$. Then $\exists q_1 \in S_t, q'_1 = h(q_1)$ such that $\forall i \in [0, n], (q_i, g, e_i, q_{1,i}) \in T_i$ since each h_i satisfies the forth property. We conclude $(q, g, e, q_1) \in T$.

We have shown that h is a timed bisimulation homomorphism from \mathcal{A} onto \mathcal{A}' . From theorem 1, we conclude that h is a timed bisimulation homomorphism from $(\mathcal{A} \upharpoonright_{<\tilde{V}}) \upharpoonright_{<^0}$ onto $(\mathcal{A}' \upharpoonright_{<\tilde{V}'}) \upharpoonright_{<^0}$, i.e. from \mathcal{N} onto \mathcal{N}' . Indeed a timed bisimulation homomorphism is naturally preserved by a timed priority restriction operator. \square

A.4 Proof of theorem 3

Timed AltaRica is a hierarchical modelling language so that each timed node can be expressed by a timed component. The timed priorities and the synchronisation are directly encoded into the resulting timed component. Let $\mathcal{N} = \langle V_F \cup C_F, E, <, \mathcal{N}_0, \dots, \mathcal{N}_n, (\tilde{V}, <\tilde{V}) \rangle$ be a timed node, we present the construction (extending the one given in []) of a timed component $\mathcal{C}_{\mathcal{N}} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, < \rangle$ which has the same semantics.

Definition 18 (Symbolic Semantics). *If \mathcal{N} is a timed node, we denote by $\mathcal{C}_{\mathcal{N}}$ the timed component constructed as follows:*

1. $\forall i = 0 \dots n$
 - (a) if \mathcal{N}_i is a timed component, then we define $\mathcal{N}'_i = \mathcal{N}_i \upharpoonright_{<^i}$ (the timed priority is taken away as defined in section 2);
 - (b) if \mathcal{N}_i is a timed node, then we define $\mathcal{N}'_i = \mathcal{C}_{\mathcal{N}_i}$;
 - (c) we denote $\mathcal{N}'_i = \langle V'_{S_i} \cup C'_{S_i}, V'_{F_i} \cup C'_{F_i}, E'_i, A'_i, M'_i, \emptyset \rangle$;
2. $V_S = V'_{S_0} \cup \dots \cup V'_{S_n}$ and $C_S = C'_{S_0} \cup \dots \cup C'_{S_n}$;
3. $A = (\exists_{i=1..n} (V'_{F_i} \cup C'_{F_i})) \cdot \bigwedge_{i=0..n} A'_i$; then by definition for $((s, \nu), (f, \mu)) \in \mathcal{D}^{V_S} \times \mathbb{R}^{C_S} \times \mathcal{D}^{V_F} \times \mathbb{R}^{C_F}$, $((s, \nu), (f, \mu)) \in \llbracket A \rrbracket \iff \exists \forall i \in [1..n], \exists \eta_i \in \mathcal{D}^{V'_{F_i}} \times \mathbb{R}^{C'_{F_i}}$ such that $(q, (f, \mu), \eta_1, \dots, \eta_n) \in \llbracket \bigwedge_{i=0..n} A'_i \rrbracket$ with $\forall i \in [1..n], ((q, (f, \mu), \eta_1, \dots, \eta_n)) \llbracket A'_i \rrbracket \iff (q_i, \eta_i) \in \llbracket A'_i \rrbracket$.
4. the set of macro-transitions $M \subseteq (\mathbb{F} \times \mathcal{B}(C_T)) \times E \times (\mathbb{E}(V_T)^{V_T} \times \mathcal{A}(C_T))$ is defined by $M = (M' \upharpoonright_{<\tilde{V}}) \upharpoonright_{<^0}$, where $<^0$ is the timed priority relation specified in definition 15, with M' defined by $((g, \gamma), e, (a, R)) \in M'$ if and only if:
 - $\forall i = 0 \dots n, \exists ((g_i, \gamma_i), e_i, (a_i, R_i)) \in M'_i, g = (\exists_{i=1..n} V_{F_i}) \cdot g_0 \wedge \dots \wedge g_n$, and $\gamma = (\exists_{i=1..n} C_{F_i}) \cdot \gamma_0 \wedge \dots \wedge \gamma_n$,
 - $\forall x \in V_S, x \in V'_{S_i} \implies a(x) = a_i(x)$ and $\forall c \in C_S, c \in C'_{S_i} \implies R(c) = R_i(c)$,
 - $e = (e_0, e_1, \dots, e_n) \in \tilde{V}$. \square

We now prove theorem 3:

Proof of theorem 3. *Let $\mathcal{N} = \langle V_F \cup C_F, E, <, \mathcal{N}_0, \dots, \mathcal{N}_n, (\tilde{V}, <\tilde{V}) \rangle$ be a timed node. We denote $\mathcal{C}_{\mathcal{N}} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, < \rangle$ its rewriting in a timed component.*

All the \mathcal{N}_i are timed components. We consider the TITS $\llbracket \mathcal{N} \rrbracket = \langle E_t, F_t, S_t^0, \pi^0, T^0 \rangle$ and $\llbracket \mathcal{C}_{\mathcal{N}} \rrbracket = \langle E_t, F_t, S_t^1, \pi^1, T^1 \rangle$ that give the semantics of \mathcal{N} and $\mathcal{C}_{\mathcal{N}}$.

First we check that $S_t^1 = S_t^0$ which amounts to checking that $\pi^1 = \pi^0$. By definition 15 $S_t^0 = \{q \in S_{0t} \times S_{1t} \cdots \times S_{nt}, \pi^0(q) \neq \emptyset\}$ with $\pi^0(q_0, q_1, \dots, q_n) = \{(f, \mu) \in \mathcal{D}^{V_F} \times \mathbb{R}^m, \forall i \in [1..n], \exists \eta_i \in \pi_i(q_i), ((f, \mu), \eta_1, \eta_2, \dots, \eta_n) \in \pi_0(q_0)\}$ with $m = |C_F|$. By the definition of the semantics of a node we have: $\pi_i(q_i) = \{\eta_i, (q_i, \eta_i) \in \llbracket A'_i \rrbracket\}$ and $\pi_0(q_0) = \{((f, \mu), \eta_1, \dots, \eta_n) \in \llbracket A'_0 \rrbracket\}$. It follows that:

$$\begin{aligned} \pi^0(q) &= \{(f, \mu) \in \mathcal{D}^{V_F} \times \mathbb{R}^m, \forall i \in [1..n], \exists \eta_i, (q_i, \eta_i) \in \llbracket A'_i \rrbracket \text{ and } (q, (f, \mu), \eta_1, \dots, \eta_n) \in \llbracket A'_0 \rrbracket\} \\ &= \{(f, \mu) \in \mathcal{D}^{V_F} \times \mathbb{R}^m, \forall i \in [1..n], (q, (f, \mu)) \in \llbracket \exists(V'_{F_i} \cup C'_{F_i}).A'_i \rrbracket \text{ and } (q, (f, \mu)) \in \\ &\quad \llbracket \exists_{i=1..n}(V'_{F_i} \cup C'_{F_i}).A'_0 \rrbracket\} \\ &= \pi^1(q) \end{aligned}$$

and then $\pi^1 = \pi^0$ and $S_t^0 = S_t^1$.

Now we prove that $T^0 = T^1$. Just notice that thanks to theorem 1, it is enough to prove that the transition relation $T_{\mathcal{N}}$ given in definition 15, is the same as the transition relation M' given in definition 18.

We first prove $T_{\mathcal{N}} \subseteq M'$. let $\langle (q_0, q_1, \dots, q_n), f, e, (q'_0, q'_1, \dots, q'_n) \rangle \in T^0$. Two cases arise:

- $e = (e_0, e_1, \dots, e_n)$ and the transition of the node is a discrete step. By definition 15, $\exists f_0 = ((f, \mu), \eta_1, \dots, \eta_n) \in \pi_0(q_0)$ and $\forall i \in [0..n], (q_i, \eta_i, e_i, q''_i) \in T_i \wedge \eta_i \in \pi_i(q_i)$. This entails that $\forall i \in [0..n], \exists ((g_i, \gamma_i), e_i, (a_i, R_i)) \in M'_i$ such that $(q_i, \eta_i) \in \llbracket A'_i \wedge g_i \wedge \gamma_i \rrbracket$ and $q''_i = (a_i, R_i)(q_i, \eta_i)$ and $(q_i, \eta_i) \in \llbracket A'_i \rrbracket$. Let us take the transition $((g, \gamma), e, (a, R))$ created in $\mathcal{C}_{\mathcal{N}}$ defined by: $g = (\exists_{i=1..n} V_{F_i}).g_0 \wedge \dots \wedge g_n$, $\gamma = (\exists_{i=1..n} C_{F_i}).\gamma_0 \wedge \dots \wedge \gamma_n$, $e = (e_0, e_1, \dots, e_n)$, (a, R) as given in definition 18. It is clear that $(q, (f, \mu), \eta_1, \dots, \eta_n) \in \llbracket A \rrbracket$. It remains to prove that $(q, (f, \mu), \eta_1, \dots, \eta_n) \in \llbracket g \wedge \gamma \rrbracket$. As $\forall i \in [1..n], (q_i, \eta_i) \in \llbracket g_i \wedge \gamma_i \rrbracket$, and $(q, (f, \mu), \eta_1, \dots, \eta_n) \in \llbracket A'_0 \rrbracket$, $(q, (f, \mu)) \in \llbracket g = (\exists_{i=1..n} V_{F_i}).g_0 \wedge \dots \wedge g_n \rrbracket$ and $(q, (f, \mu)) \in \llbracket \gamma \rrbracket$. The same follows straightforwardly for γ . Also as (a, R) agrees with the a_i, R_i on each sets $V'_{S_i} \cup C'_{S_i}$, the new states q''_i are equal to the q'_i . Also as $(e_0, e_1, \dots, e_n) \in \tilde{V}$, this synchronised vector is in M' .
- $e = \delta \in \mathbb{T}$ and the node does a time step. The proof is exactly the same as above: the updates are just time updates where the values of the clocks increase of δ time units.

The converse $M' \subseteq T_{\mathcal{N}}$ is quite easy to prove along the same lines.

General case. The induction assumption is here that all the \mathcal{N}_i can be rewritten as timed components \mathcal{N}'_i : then the previous result applies (the only thing needed is to use theorem 2). This ends the proof of the theorem. \square

A.5 Proof of theorem 4

In the section 4, we propose a procedure to translate a timed component into a timed automaton.

Proof of theorem 4. We are going to define $\llbracket \mathcal{A}_{\mathcal{T}} \rrbracket$ by a TITS. The set of state variables coincides with the discrete variables and the clocks which are modified by any transition. $\llbracket \mathcal{A}_{\mathcal{T}} \rrbracket = \langle E_t, F_t^1, S_t^1, \pi^1, T^1 \rangle$ with $F_t^1 = \bigcup \text{proj}_{F_t} \llbracket l_T^F \rrbracket$, $S_t^1 = \bigcup \text{proj}_{S_t} \llbracket l_T^F \rrbracket$. Then each element of F_t^1 and S_t^1 are a subset of respectively F_t and S_t . Indeed, $\forall G \in F_t^1, G = \text{proj}_{F_t}(\llbracket l_T^F \rrbracket) = \{g \in F_t, g \in \text{proj}_{F_t}(\llbracket l_T^F \rrbracket)\} \in 2^{F_t}$. It is the same for all $Q \in S_t^1, Q = \text{proj}_{S_t}(\llbracket l_T^F \rrbracket)$. We next define π^1 .

For all $q, q' \in \text{proj}_S(l_T^F)$, we have $\pi(q) = \pi(q')$. So that, we can write $\pi(l_T^F)$. For this, we show that $\bigwedge_{j=1}^p P_j \implies I_j$ is equal to $\bigwedge(\bigwedge_{j \in T} P_j) \bigwedge(\bigwedge_{j \in F} \neg P_j) \implies \bigwedge_{k \in T} I_k$. We do it by induction.

First, $P \implies I$ is rewriting in $P \implies I \wedge \neg P \implies \text{true}$

Second, assume the hypothesis for p conditions. We consider

$$\begin{aligned}
(1) \quad & \bigwedge_{j=1}^p P_j \implies I_j \wedge P_{p+1} \implies I_{p+1} \\
& \Leftrightarrow \bigwedge_{j=1}^p (P_j \wedge P_{p+1} \implies I_j \wedge I_{p+1}) \wedge \bigwedge_{j=1}^p (P_j \wedge \neg P_{p+1} \implies I_j) \wedge \\
& \quad (\bigwedge_{j=1}^p \neg P_j \wedge \neg P_{p+1} \implies \text{true}) \\
& \quad \text{we put } P_i \wedge P_{p+1} = P'_i \text{ and } P_i \wedge \neg P_{p+1} = P''_i \\
& \Leftrightarrow \bigwedge(\bigwedge_{j \in T} P'_j) \bigwedge(\bigwedge_{j \in F} \neg P'_j) \implies \bigwedge_{k \in T} I_k \wedge I_{p+1} \wedge \\
& \quad \bigwedge(\bigwedge_{j \in T} P''_j) \bigwedge(\bigwedge_{j \in F} \neg P''_j) \implies \bigwedge_{k \in T} I_k \wedge (\bigwedge_{j=1}^p \neg P_j \wedge \neg P_{p+1} \implies \text{true}) \\
& \Leftrightarrow \bigwedge(\bigwedge_{j \in T} P_j \wedge P_{p+1}) \bigwedge(\bigwedge_{j \in F} \neg(P_j \wedge P_{p+1})) \implies \bigwedge_{k \in T} I_k \wedge I_{p+1} \wedge \\
& \quad \bigwedge(\bigwedge_{j \in T} P_j \wedge \neg P_{p+1}) \bigwedge(\bigwedge_{j \in F} \neg(P_j \wedge \neg P_{p+1})) \implies \bigwedge_{k \in T} I_k \wedge \\
& \quad (\bigwedge_{j=1}^p \neg P_j \wedge \neg P_{p+1} \implies \text{true}) \\
& \Leftrightarrow \bigwedge(\bigwedge_{j \in T} P_j \wedge P_{p+1}) \bigwedge(\bigwedge_{j \in F} \neg(P_j)) \implies \bigwedge_{k \in T} I_k \wedge I_{p+1} \wedge \\
& \quad \bigwedge(\bigwedge_{j \in T} P_j \wedge \neg P_{p+1}) \bigwedge(\bigwedge_{j \in F} \neg P_j) \implies \bigwedge_{k \in T} I_k \wedge \\
& \quad (\bigwedge_{j=1}^p \neg P_j \wedge \neg P_{p+1} \implies \text{true}) \vee \text{false}
\end{aligned}$$

Hence, we have the result. Then, let $q, q' \in \text{proj}_{V_S}(\llbracket l_T^F \rrbracket)$,

$$\begin{aligned}
\pi(q) &= \{g, (q, g) \in \llbracket A \rrbracket\} \\
& \quad \{g, (q, g) \in \llbracket A_{V_T} \rrbracket \cap \llbracket A_{C_T} \rrbracket\} \\
& \quad \{g, (q, g) \in \llbracket A_{V_T} \rrbracket \cap \llbracket r_T^F \implies \bigwedge_{k \in T} I_k \rrbracket\} \\
& \quad \{g, (q, g) \in \bigcap \llbracket l_T^F \rrbracket \cap \llbracket \bigwedge_{k \in T} I_k \rrbracket\} \\
& \quad \{g, (q, g) \in \llbracket l_T^F \rrbracket \cap \llbracket \bigwedge_{k \in T} I_k \rrbracket\} \\
&= \pi(q')
\end{aligned}$$

Finally, the transition relation T^1 is defined by:

$\llbracket (l_T^F, (g \wedge \text{Pre}_t(l_{T'}^{F'}) \wedge \gamma, e, (a, R)), l_{T'}^{F'}) \rrbracket = (Q, G, e, Q') \in T^1$ is such that $Q = \text{proj}_{S_t}(\llbracket l_T^F \rrbracket)$, $G = \text{proj}_{F_t}(\llbracket l_T^F \rrbracket)$ and $Q' = \text{proj}_{S_t}(\llbracket l_{T'}^{F'} \rrbracket)$. By definition of the transition relation in $\mathcal{A}_{\mathcal{T}}$, we know that $Q \in \llbracket g \wedge \text{Pre}_t(l_{T'}^{F'}) \rrbracket$ and $(a, R)(Q, G) = Q'$.

Once $\llbracket \mathcal{A}_{\mathcal{T}} \rrbracket = \langle E_t, F_t^1, S_t^1, \pi, T^1 \rangle$ defined, we can show that this TITS is timed bisimilar to $\llbracket \mathcal{T} \rrbracket = \langle E_t, F_t, S_t, \pi, T \rangle$. Precisely, we show that $\forall q, Q \in S_t \times S_t^1$, $q \sim Q \Leftrightarrow q \in Q$. Using the definition 16, since a state $Q \in S_t^1$ is a grouping of elements of S_t , the first point is trivial. The second point comes from the definition of π^1 . Concerning the transition relation, for a transition (Q, G, e, Q') we just take a valuation $q, q', g \in Q^2 \times G$, then $(q, g, e, q') \in T$. Conversely for a transition $(q, g, e, q') \in T$, we take the classes associated to $q, g, q' \in Q^2 \times G$ to obtain the transition $(Q, G, e, Q') \in T^1$.

□