A timed extension for AltaRica

Franck Cassez, Claire Pagetti, Olivier Roux

Fundamenta Informaticae XXI (2001) 1001–1042 1001
IOS Press

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

Abstract. In this paper we present a *timed* extension of the AltaRica formalism. Following previous works, we first extend the semantics of AltaRica with time and define *timed components* and *timed nodes*. Moreover we lift the *priority features* of AltaRica to the timed case. We obtain a timed version of AltaRica, called Timed AltaRica. Finally we give a translation of a Timed AltaRica specification into a usual timed automaton. These are the semantic foundations of a high-level hierarchical language for the specification of timed systems.

Keywords: AltaRica, Semantics, Timed Automata

1. Introduction

Context. The development of complex and safety-critical systems requires the use of formal methods and tools for system design and specification. In the case of *discrete* systems the so-called *reactive languages* [1, 2, 3, 4] have been used for almost a decade to specify industrial systems. They give a rigourous and elegant basis for the structured development of reactive systems with the use of *composition* and *hierarchical* specifications for instance. On those specifications such techniques like *model-checking* can be applied to check for some properties on the designed systems.

The need for a counterpart specification language in the case of *timed specifications* arose recently as timing information can now be dealt with while verifying a system with tools like UPPAAL [5], CMC [6], KRONOS [7] or HyTech [8]. We give here the theoretical foundations of such a high-level

Address for correspondence: Franck Cassez, IRCCyN, BP 92101, 1 rue de la Noë, 44321 Nantes Cedex 3, France

specification language for timed systems. We extend the AltaRica [9, 10, 11] formalism with *timing* features.

AltaRica is a high-level specification formalism that allows one to specify *constraint automata* [9] 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 synchronization mechanism. Such a general component is called a *node*. One can express broadcast communication, give priority among some transitions, etc.

Moreover AltaRica has an unambiguous semantics [11, 10] 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-trees to perform reliability analysis [12], Petri nets, Markov graphs or finite state automata (that can be analysed with the tool MEC [13, 14, 15, 16] for instance).

Nevertheless one cannot specify real-time constraints in AltaRica and of course this becomes crucial when some timing information contributes to the modelling and correctness 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* [17] and *hybrid automata* [18, 19] and many efficient tools are now available [8, 7, 20]. This adds a new feature to the AltaRica toolbox.

Our Contribution. Our work consists in extending the AltaRica formalism with *real-time constraints* and define a timed version of AltaRica called Timed AltaRica. We thus extend the theoretical foundations of AltaRica: we enhance 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 theoretical results obtained for AltaRica (e.g. interface bisimulation homomorphism, rewriting of a node into a component, ...) to the timed case: this is important as it gives Timed AltaRica *good* compositional properties that are needed in practice. Finally we present an algorithm to compile Timed AltaRica specifications into timed automata (which can be then analyzed with UPPAAL [5]).

Outline of the paper. In the next section, we remind the basics about AltaRica and introduce a running example: the Train-Gate-Controller example. Section 3 is the core of the paper and presents Timed AltaRica 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 specifications. We conclude by some perspectives in section 6.

The proofs of the theorems are given in the appendices (pages 1035– 1042).

2. An Overview of the AltaRica Language

In this section we recall some basics of AltaRica [10, 11] and give an example of an AltaRica specification:the train-gate-controller [21]. In this example, the aim is to keep the gate closed when a train is in

a critical section. We will use three AltaRica components to model the system and describe how they synchronize.

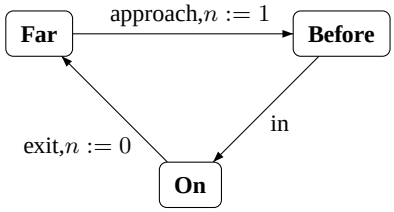
2.1. Specifying Reactive Systems in AltaRica

A specification in AltaRica is a *node*. A node is a *hierarchical* description. It can be built from sub-nodes and so on. A node that contains no sub-nodes is a *component*. A node is basically composed of:

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

```
1:  node TRAIT
2:    flow   N : [0,1];      // These are the flow variables
3:    event  approach, in, exit;
4:    state  etat : [0,2]; n : [0,1];
5:    trans
6:        etat=0 |- approach ->
7:                    etat := 1, n := 1;
8:        etat=1 |- in -> etat := 2;
9:        etat=2 |- exit -> etat := 0, n := 0;
10:   init
11:       etat:=0, n:=0;
12:   assert
13:       N=n;
14: edon
```

(a) Spec. of the Train in AltaRica



$$\begin{aligned} \textit{Far} &\equiv etat = 0 \\ \textit{Before} &\equiv etat = 1 \\ \textit{On} &\equiv etat = 2 \end{aligned}$$

(b) Spec. of the Train as an Automaton

Figure 1. Specification of a Train

2.1.1. Components

In the example of Fig. 1, we define a component[1] *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)). 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` (line 4) ranging in $[0, 2]$ represents the locations *Far*, *Before*, *On* of the train. The events of the component `TRAIN` are *approach*, *in* and *exit* (line 3). We also use a state variable `n` (line 4) to denote that the train is in $\{\textit{Before}, \textit{On}\}$. Initially the component is in configuration `etat=0,n=0,N=0` (line 11), written $(0, 0, 0)$ for short. When a transition occurs the values of the state variables change accordingly as well as the

[1] In AltaRica the keyword `node` in used for components (nodes with no sub-nodes) as well as for hierarchical nodes; indeed a component is a special case of node with no sub-nodes.

value of the flow variable in order to satisfy the assertion (line 13). For instance, when event *approach* occurs in $(0,0,0)$ the configuration $(1,1,1)$ is reached.

2.1.2. Interfaces

The component's state variables are not visible from outside of the component. Their scope is thus the component itself just as for usual programming languages. To allow sharing of information and synchronization on variables of other components one can use *flow* variables. Flow variables can be read by other nodes. The part of the component which is visible by other components is called the *interface*. It consists in the events of the components and the flow variables.

The flow variable `N` is in the *interface* of the node `TRAIN` (line 2). This means that other nodes can read it and use the value of `N`. The value of the flow variable `N` is constrained to be equal to `n` at anytime (see the `assert` line 13) and the purpose of `N` is to make the value of `n` available outside.

Assume another node for the controller is given by the AltaRica specification of Fig. 2(a). A transition of the form `etat = 1 |- approach -> ;` (line 9) means that `approach` does not bring about any change in the state variables values (but not this is a deadlock!). In the component `CONTROLLER` the purpose of the flow variable `N` (referred to as `CONTROLLER.N` from now on) is to count the total number of trains in the region $\{\textit{Before}, \textit{On}\}$ (if we assume there are many train components). Depending on the value of the flow `CONTROLLER.N` the controller will make the gate go up on an *exit* signal (if the value is 1, line 8) or will leave the gate closed if `CONTROLLER.N` > 1 (line 7).

The value of `CONTROLLER.N` may change on any discrete transition and be assigned any integer as no assertion constrains this flow in the node `CONTROLLER`. Apart from the events listed in the component's events section (line 3), we assume a special discrete event ϵ for synchronization purposes. This event is enabled in any configuration and does not change the values of variables of type `state`. Nevertheless flow variables can be updated on ϵ transitions with values satisfying the assertion. As the assertion of the node `CONTROLLER` is implicitely `true` the variable `CONTROLLER.N` may be assigned any integer value on an ϵ transition. This somewhat strange behavior will become clear when we introduce hierarchical nodes and constraints among flows of different nodes (see `assert` line $\ell_{main-assert}$ on Fig. 3).

As for the node `GATE` (Fig. 2(b)) it consists in receiving orders from the controller (events `Go_up` and `Go_down`) and after a while[2] to actually go up or down (events `up` and `down`).

2.1.3. Hierarchy and Synchronization

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. 3) specifying the train-gate-controller with two trains. Indeed nodes can be instantiated and used as templates to build higher-level nodes. The node of Fig. 3 is composed of four instantiated sub-nodes (`t1`, `t2`, `g` and `c`, see Fig. 3, lines 2–5) which interact in two ways: flow coordination and synchronization of events. The synchronization constraint (after keyword `sync`, lines 6–14) reads as follows: if a component does not appear in a synchronization vector, it is assumed to do the ϵ action. Note that events `up` and `down` are not synchronized and thus they will be assumed to be synchronized with ϵ transitions of the other components. Finally the global assertion, line 15, constrains the flow variables so that `N` of the node `MAIN` is always equal

[2] We will see later how this can be made precise using timing constraints to make a timed version of the controller and the gate in Fig. 5.

```
1:  node CONTROLLER
2:    flow  N : [0,p];
3:    event approach, exit, Go_up, Go_down;
4:    state etat : [0,2];
5:    trans
6:      etat=0 |- approach -> etat := 1;
7:      etat=0 & N>1 |- exit -> ;
8:      etat=0 & N=1 |- exit -> etat := 2;
9:      etat = 1 |- approach -> ;
10:     etat = 1 |- exit -> ;
11:     etat=1 |- Go_down -> etat := 0;
12:     etat=2 |- Go_up -> etat := 0;
13:     etat=2 |- approach -> etat := 1;
14:   init  etat := 0, z := 0;
15: edon
```

(a) The Controller

```
node GATE
 event Go_down, Go_up, down, up;
 state  etat : [0,3];
 trans
   etat=0 |- Go_up -> ;
   etat=0 |- Go_down -> etat := 1;
   etat=1 |- Go_down -> ;
   etat=1 |- down -> etat := 2;
   etat=1 |- Go_up -> etat :=3;
   etat=2 |- Go_down -> ;
   etat=2 |- Go_up -> etat := 3;
   etat=3 |- Go_up -> ;
   etat=3 |- Go_down -> etat := 1;
   etat=3 |- up -> etat := 0;
 init    etat:=0;
edon
```

(b) The Gate

Figure 2. AltaRica Specifications for the Controller and the Gate

to the number of trains on the critical section. A joint move of the components `t1,t2,g,c` can be `<t1.approach,t2.approach,c.approach>` (see line 7) in which case the variable `c.N` will be updated on the ϵ move of component `c` to satisfy the assertion of node `MAIN` *i.e.* `c.N=t1.N+t2.N`. This is why we need to have the possibility to update flow variables on ϵ transitions. Anyway a meaningful specification should be such that all flow variables are constrained at least in the outermost node. Note that some constraints could be unsatisfiable: for instance if we add `t1.N=2+t2.N` to the `assert` line, this clearly can not be satisfied and the resulting system has no configuration. It is also possible to constrain the state space: if we use `t1.N=t2.N` we impose that the two trains issue `approach` at the same time and leave the critical section at the same time (event `exit`). This is due to the fact that no configuration with `t1.N` not equal to `t2.N` is satisfiable hence no transition with only one `approach` event can be fired.

```
1:  node MAIN
2:  sub
3:    t1,t2 : TRAIN;
4:    g : GATE;
5:    c : CONTROLLER;
6:  sync
7:    <t1.approach,t2.approach,c.approach>;
8:    <t1.approach,c.approach>;
9:    <t2.approach,c.approach>;
10:   <t1.exit,t2.exit,c.exit>;
11:   <t1.exit,c.exit>;
12:   <t2.exit,c.exit>;
13:   <g.Go_down,c.Go_down>;
14:   <g.Go_up,c.Go_up>;
15: assert  c.N=t1.N+t2.N;
16: edon
```

Figure 3. Hierarchical Node

2.2. 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 [11, 10].

2.2.1. Interfaced Transition Systems

Definition 2.1. (Interfaced Transition system [10])
An *interfaced transition system* (ITS) is a tuple $\mathcal{A} = \langle E, F, S, \pi, T \rangle$ with:

1. $E = E_+ \cup \{\varepsilon\}$ is a finite set of *events* such that $\varepsilon \notin E_+$;
2. F is a set of *flow* values;
3. S is the set of *states*;
4. $\pi : S \to 2^F$ 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) $\forall s \in S, \forall f \in \pi(s),\ (s, f, \varepsilon, s) \in T$

A *configuration* of an ITS is a pair $(s, f) \in S \times F$ such that $f \in \pi(s)$. Every tuple $(s, f, e, s') \in T$ corresponds to the set of transitions $((s, f), e, (s', f'))$ between configurations s.t. $f' \in \pi(s')$.

Remark 2.1. In AltaRica, if a transition (s, f, e, s') is firable then there exists a configuration (s', f') (as item 4 of Def. 2.1 assumes $\pi(s)$ is not empty for $s \in S$). 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. Also note that T is a shorthand for the explicit transition relation T' between configurations with $T' \subseteq S \times F \times E \times S \times F$ and $(s, f, e, s', f') \in T' \iff (s, f, e, s') \in T \wedge f' \in \pi(s')$.

2.2.2. 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 [22]. 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 2.2. (Priority relation [10])
A *priority relation* over E is a strict partial order over E such that $\forall v \in E_+, v \not< \varepsilon$ and $\varepsilon \not< v$ (with $E_+ = E \setminus \{\varepsilon\}$).

Definition 2.3. (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* $\restriction$ for the transition relation $T \subseteq S \times F \times E \times S$ and the priority relation $<$ by: $(s, f, e, s') \in T\restriction_< \iff (s, f, e, s') \in T \wedge \big(\forall e' \in E\,,\ (\exists s' \in S \,|\, (s, f, e', s') \in T) \implies e \not< e'\big)$.

2.2.3. 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 $\nu : X \to \mathcal{D}$ 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 $\nu : X \to \mathcal{D}$ 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 (true) and ff (false) 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 $[\![f]\!]$ of a formula $f \in \mathbb{F}(X)$ with *free*$(f) \subseteq X'$ is a subset of $\mathcal{D}^{X'}$: $[\![f]\!] \subseteq \mathcal{D}^{X'}$. Also we have $[\![tt]\!] = \mathcal{D}^X$ and $[\![ff]\!] = \emptyset$.

An *assignment* for the variables in X is a mapping $a : X \to \mathbb{E}(X)$. Intuitively an assignment of the form $x := y + z + 2$ will be defined by $a(x) = y + z + 2$. Given a valuation $\nu : X \to \mathcal{D}$, 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 [10].

2.2.4. AltaRica Components

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

Definition 2.4. (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 $V_T = V_S \cup V_F$;
2. $E = E_+ \cup \{\varepsilon\}$ 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 $(tt, \varepsilon, Id) \in M$ and every $(g, e, a) \in M$ satisfies:
   (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) $a : V_S \to \mathbb{E}(V_C)$ is an *assignment* for the variables in V_S,
5. $<$ is a priority relation.

Remark 2.2. In [10], 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 in $V_S \cup V_F$ have a common domain $\mathcal{D}$.

Definition 2.5. (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 $[\![\mathcal{C}]\!] = \langle E, F, S, \pi, T \rangle$ constructed in the following way:

1. $F = \mathcal{D}^{V_F}$;
2. $S = \{s \in \mathcal{D}^{V_S} \mid \exists f \in \mathcal{D}^{V_F} \mid \ (s, f) \in [\![A]\!]\}$;
3. $\pi : S \to 2^F$ such that $\pi(s) = \{f \mid (s, f) \in [\![A]\!]\}$;
4. $T \subseteq S \times F \times E \times S$ is given by $T = [\![M]\!] \restriction <$ with:
   (a) let $t = (g, e, a) \in M$, then $[\![t]\!] = \{(s, f, e, s') \mid (s, f) \in [\![A \wedge g]\!] \wedge s' = a(s, f)\}$,
   (b) $[\![M]\!] = \cup_{t \in M} [\![t]\!]$.

Note that because of item 4 above the requirement $\pi(s) \neq \emptyset$ for ITS is always fulfilled.

2.2.5. AltaRica Nodes

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

Definition 2.6. (Node)
A *node* is a tuple $\mathcal{N} = \langle V_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, V \rangle$ with:

1. V_F is a set of *flow* variables,
2. $E = E_+ \cup \{\varepsilon\}$ 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 $\mathcal{N}_0$ 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 $V_{F_0} = V_F \cup V_{F_1} \cup V_{F_2} \cup \cdots \cup V_{F_n}$,
6. $V = V_d \cup V_{imp}$ is the set of *specified* synchronization vectors:
   - $V_d \subseteq E_0^? \times \cdots \times E_n^? \times 2^{[0,n+1]}$ where $E_i^? = E_i \cup \{?e | e \in E_{i+}\}$; we define E_d^i by: $e \in E_d^i$ if $\exists \langle \cdots, x_i, \cdots \rangle \in V_d$ with $x_i \in E_i^?$; E_d^i corresponds to the set of events of node i that are synchronized; V_d induces a set of synchronization vectors (see below). The last component in $2^{[0,n+1]}$ constrains the sets of "?"-events in the nodes that need to participate in the synchronization (see below).
   - $V_{imp} \subseteq E_0 \times \cdots \times E_n \times \{\emptyset\}$ is the set of *implicit* synchronization vectors with:

– $\langle \varepsilon, \cdots, \varepsilon, \emptyset \rangle \in V_{imp}$,
– $\forall i \in [0, n], \forall e_i \in E_i \setminus E_d^i$, $\langle \varepsilon, \cdots, e_i, \cdots, \varepsilon, \emptyset \rangle \in V_{imp}$.

V_{imp} contains all the synchronization vectors with non synchronized events.

An example of how V_d generates synchronization vectors can be given by the node `MAIN` of Fig. 3. Assume in this node, we replace the line 7by `<t1.approach?,t2.approach?,c.approach> >= 1`. The meaning of this new specification is that it induces the set of synchronization vectors in which more than 1 (given by the `>=1` constraint) event qualified with a "?" appears. Thus `<c.approach>` is not an allowed vector whereas `<t1.approach,c.approach>`, `<t2.approach,c.approach>` and `<t1.approach,t2.approach,c.approach>` are allowed. The *unfolding* of the following constrained vector `<t1.approach?,t2.approach?,c.approach> >= 1` contains only the three allowed vectors defined above. Note that our definition involving subsets of $[0, n+1]$ allows us to specify more precise vectors than the one given by the number of "?"-events that have to be present. The synchronization set V generates a set of synchronization vectors of $E_0 \times E_1 \times \cdots \times E_n$ together with a priority relation on them[3]. As already mentioned, a vector of the form `<t1.approach?,t2.approach?,c.approach> >= 1` generates all the synchronization vectors containing at least one event the name of which is qualified by a "?". The priority relation for those vectors corresponds to giving priority to the one with the maximal number of "?"-events occurring in the vector: in the previous case `<t1.approach,c.approach>` and `<t2.approach,c.approach>` are both strictly lower (have less priority) than the 3-component vector `<t1.approach,t2.approach,c.approach>`. In this case, each time both `t1.approach` and `t2.approach` are simultaneously enabled this priority relation imposes they are fired at the same time. Thus this specification rules out the behaviours where only one of these transitions is fired whereas the other is enabled. We do not want to constraint the system in such a way and `approach` events cannot be constrained in the specification. This is why we have given three distinct synchronization vectors involving event `approach` and they are independant from each other.

Finally, the set V_{imp} consists of all the events that are not involved in any synchronization: they must occur on their own, hence the synchronization vectors of the form $\langle \varepsilon, \cdots, e, \cdots, \varepsilon \rangle$ (events `up` and `down` of components `Gate` of Fig. 2(b)).

For a formal definition of how to generate the synchronization vectors corresponding to V the reader is referred to [10]. We only need here to consider the set of synchronization 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 synchronization vectors $\widetilde{V}$ it generates and the priority relation $<_{\widetilde{V}}$ it induces, *i.e.* we will use $\widetilde{V} \subseteq E_0 \times E_1 \times \cdots \times E_n$ and $<_{\widetilde{V}}$ instead of V.

There is a fundamental result about nodes: they can be rewritten (syntactically) into components that preserve their semantics [10].

Theorem 2.1. ([10, 11])
If $\mathcal{N}$ is an AltaRica node, $\mathcal{C}_{\mathcal{N}}$ its rewriting into a component (as defined in [10]), then $[\![\mathcal{N}]\!]$ and $[\![\mathcal{C}_{\mathcal{N}}]\!]$ are bisimilar.

In the next section we focus on extending ITS, 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 [10, 11]

[3] how this is done is defined in [10].

still hold (e.g. Theorem 2.1).

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. 3.1) and define *timed* priorities. Then we add timing constraints 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 [23]. More precisely we use the framework of timed automata [17] 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. A *clock assignment* is a mapping $a : X \to \mathbb{E}(X)$. For decidability reasons, we will restrict the allowed assignment expressions in section 4.4 to simple assignments given by table 2, page 1030. We denote by $\mathcal{A}(X)$ the set of clock assignments. As defined in subsection 2.2.3, 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:

$$g := x \backsim r | \; x - y \backsim r \; | g \wedge g \; | g \vee g \tag{1}$$

with $x, y \in X, \backsim \in \{<, <, >, \geq, =\}$, $r \in \mathbb{Q}$. 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: $[\![g]\!] \subseteq \mathbb{R}^X_{\geq 0}$ and $g(\nu) = tt \iff \nu \in [\![g]\!]$.

Timed Transition Systems and Timed Automata. A *timed transition system* [23] (TTS) is a tuple $(Q, E, Q_0, \to)$, where Q is set of locations, E is the set of actions, Q_0 is the set of initial states, $\to \subseteq Q \times (E \cup \mathbb{R}_{\geq 0}) \times Q$. A *timed automaton* [17] is a tuple (L, L_0, E, X, I, T) such that L is a (finite) set of *locations*, X is a finite set of *clocks*, L_0 is s.t.$[\![L_0]\!] \subseteq L \times \mathbb{R}^X_{\geq 0}$ is a predicate that defines the set of *initial* states, E is a finite set of *actions*, $T \subseteq L \times (\mathcal{B}(X) \times E \times \mathcal{A}(X)) \times L$ is the *transition relation*, $I : L \to \mathcal{B}_C(X)$ is the *invariant* constraint.

The semantics of a timed automaton (L, L_0, E, X, I, T) is given by a TTS $(L \times \mathbb{R}^X_{\geq 0}, E, Q_0, \to)$ where $Q_0 = [\![L_0]\!]$ and $\forall (l, v) \in L \times \mathbb{R}^X_{\geq 0}$ the transition relation $\to$ is defined by: i) *discrete* steps of the form $(l, v) \xrightarrow{e} (l', v')$ if $\exists (l, g, e, a, l') \in T$, *such that* $g(v) = tt, v' = a(v), v' \in [\![I(l')]\!]$, ii) *continuous* steps of the form $(l, v) \xrightarrow{\delta} (l, v'), \delta \in \mathbb{R}_{\geq 0}$ if $\forall \delta' \leq \delta, v + \delta' \in [\![I(l)]\!]$.

A very useful result about timed automata (actually *updatable* timed automata [24]) is that reachability is decidable [17, 24] for this class of timed systems. Hence automatic verification tools have been designed to analyse timed automata, and among them UPPAAL [5], KRONOS [7] and CMC [6]. 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 [5] or KRONOS [7] or CMC [6] 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 3.1. (Timed Interfaced Transition System)
A *timed interfaced transition system* (in the sequel TITS) of *continuous dimension* (n, m) and *time domain*[4] $\mathbb{T}$ is a tuple $\mathcal{A} = \langle E_t, F_t, S_t, \pi, T \rangle$ with:

1. $E_t = E_+ \cup \{\varepsilon\} \cup \mathbb{T}$ where E_+ is a finite set of *events* such that $\varepsilon \notin E_+ \cup \mathbb{T}$ and $E_+ \cap \mathbb{T} = \emptyset$;
2. $F_t \subseteq F \times \mathbb{R}^m$ 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. $S_t \subseteq S \times \mathbb{R}^n$ is the set of *states* where S is the set of *discrete states* and $\mathbb{R}^n$ is the set of *continuous states*;
4. $\pi : S_t \to 2^{F_t}$ 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. $T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ is the *transition relation* and satisfies:
   (a) $(q, g, e, q', g') \in T \Rightarrow g \in \pi(q) \wedge g' \in \pi(q')$
   (b) $\forall q \in S_t, \forall g, g' \in \pi(q)$ we have $(q, g, \varepsilon, q, g') \in T$
   (c) $\forall q \in S_t, \forall g \in \pi(q)$ we have $(q, g, 0, q, g) \in T$

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 3.1. Compared to item 5 of Def. 2.1, we need to be more precise when defining the set of transitions of a TITS. Indeed we want to enforce discrete variables to remain unchanged when time elapses. Assume we define the transition relation T in the same way as it was in defined item 5 of Def. 2.1: $T \subseteq S_t \times F_t \times E_t \times S_t$. Then we will not be able to leave discrete flow values unchanged during a delay transition when we define the semantics of AltaRica timed components (Def. 3.7): in the target configuration, we can only constrain the state variables in S_t and not the flow variables in F_t. Thus we prefer to define T over $S_t \times F_t \times E_t \times S_t \times F_t$ which enables us to refer to the source and target flow values of a transition as in item 5.(c) of Def. 3.1.

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

Also note the following properties: if $n = 0$ and $m = 0$ and $\mathbb{T} = \{0\}$ we obtain the definition of ITS. Indeed as pointed out in remark 2.1, we can give the definition of the transition relation of an ITS in terms of transitions between configurations. 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 $\mathbb{T} = \mathbb{R}_{\geq 0}$. In the following we assume the time domain is $\mathbb{R}_{\geq 0}$ when we deal with Timed AltaRica nodes.

3.3. Timed Bisimulations

In the sequel we will use the notion of *timed bisimulations* between timed systems. We define it for TITS extending the definition of *interfaced bisimulation* of [10]:

Definition 3.2. (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 4 conditions:

1. $\forall q_1 \in S_{1_t}, \exists q_2 \in S_{2_t}$ s.t. $(q_1, q_2) \in R$ and $\forall q_2 \in S_{2_t}, \exists q_1 \in S_{1_t}$ s.t. $(q_1, q_2) \in R$,
2. $\forall (q_1, q_2) \in R,\ \pi_1(q_1) = \pi_2(q_2)$,
3. $\forall (q_1, g, e, q_1', g_1') \in T_1,\ \forall q_2 \in S_{2_t}$ such that $(q_1, q_2) \in R$ then $\exists (q_2, g, e, q_2', g_2') \in T_2$ s.t. $(q_1', q_2') \in R$,
4. $\forall (q_2, g, e, q_2', g_2') \in T_2,\ \forall q_1 \in S_{1_t}$ such that $(q_1, q_2) \in R$ then $\exists (q_1, g, e, q_1', g_1') \in T_1$ s.t. $(q_1', q_2') \in R$.

Two TITS are *timed bisimilar* iff there exists a timed interfaced bisimulation relation on their set of states.

In the sequel we use the term timed bisimulation instead of timed interfaced bisimulation. Like in the untimed case, an interfaced bisimulation can be expressed as an homomorphism between two TITS.

Definition 3.3. (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* $h : \mathcal{A}_1 \to \mathcal{A}_2$ is a mapping $h : S_{1_t} \to S_{2_t}$ such that:

1. h is surjective,
2. $\forall q_1 \in S_{1_t},\ \pi_1(q_1) = \pi_2(h(q_1))$,
3. $\forall (q_1, g, e, q_1', g_1') \in T_1, \exists g_2'$ such that $(h(q_1), g, e, h(q_1'), g_2') \in T_2$,
4. $\forall q_1 \in S_{1_t}, \forall q_2' \in S_{2_t}$ s.t. $(h(q_1), g, e, q_2', g_2') \in T_2$ then $\exists q_1' \in S_{1_t}$ such that $h(q_1') = q_2'$ and $\exists g_1' \in \pi_1(q_1')$ such that $(q_1, g, e, q_1', g_1') \in T_1$.

We use the term timed bisimulation homomorphism as a shorthand for timed interfaced bisimulation homomorphism.

The following theorem is an extension to TITS of previous results on ITS and follows from [10, 25]:

Theorem 3.1. Two TITS $\mathcal{A}_1$ and $\mathcal{A}_2$ are timed bisimilar if and only if there exists a TITS $\mathcal{B}$ and two timed interfaced bisimulation homomorphisms $h_1 : \mathcal{A}_1 \to \mathcal{B}$ and $h_2 : \mathcal{A}_2 \to \mathcal{B}$.

The proof is given in Appendix A.1.

3.4. Timed Priorities

In the untimed version of AltaRica, priorities among events play an important role [10]: they allow the easy modeling 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 [26] and a comprehensive study of timed priorities can be found in [27, 28, 29]. The most common timed priority is *urgency* [30]: 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 a Timed AltaRica specification is *urgent* then all the transitions labelled by this event are urgent. We then extend Def. 2.2 to allow priority between time labels (in $\mathbb{T}$) and discrete events:

Definition 3.4. (Simple Timed Priority Relation)
A *simple timed priority relation* $<$ over E is a strict partial order over $E \cup \{\textit{time}\}$ such that $<$ is a priority relation over E and a strict partial order over $E_+^{\textit{time}}$ with $E_+^{\textit{time}} = E_+ \cup \{\textit{time}\}$ and $\forall v \in E_+, v \not< \textit{time}$.

Then $a > \textit{time}$ means[5] that event a is *urgent* and has to be fired immediately when enabled (a semantic definition of priorities will appear later in this section in Def. 3.6). Also note that if $a > \textit{time}$ and $b > a$ then $b > \textit{time}$: the urgency of event a entails urgency of greater events.

This allows us to model what is called *eagerness* in [30]: an *eager* transition is one that forbids time elapsing if it is enabled. In the papers [27, 28] 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 Def. 3.7 *(time) guards* into Timed AltaRica components which will enable us to define *lazy* transitions. It is proved in [28] that a delayable transition can be encoded using lazy an eager transitions. As we already know (Def. 3.4) how to define eager transitions, we are able to express the three types of priorities proposed in [30].

More elaborate ways of prioritising transitions are given in [27]. 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 3.5. (Timed Priority Relation)
A *timed priority relation* $<$ over E is a 3-ary relation in $E_+^{\textit{time}} \times (\mathbb{N} \cup \{\infty\}) \times E_+^{\textit{time}}$ 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,

[5] We rule out $\textit{time} > a$ as the purpose of a priority relation is to add a sort of liveness in the system by forcing some discrete actions to be taken.

- $a <_k b \wedge a = \textit{time} \Longrightarrow k = 0$,
- $\forall k \in \mathbb{N}, <_k$ is a strict partial order,
- $a_1 <_k a_2 \Longrightarrow \big(\forall k' < k, a_1 <_{k'} a_2\big)$.

Remark 3.2. Notice that in [27], 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. It is aimed at preserving liveness in the systems and can then add new behaviors. We do not want to build live timed systems but only to provide a restriction operator (by giving priorities) that restricts the set of behaviors of the system. 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 2.2.2, we define the timed priority restriction operator.

Definition 3.6. (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* $\restriction$ for the transition relation $T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ and the timed priority relation $<$ by:

$$(q,g,e,q',g') \in T\restriction\!< \Leftrightarrow (q,g,e,q',g') \in T \wedge \begin{cases} \textit{if } e = t \in \mathbb{T}, \forall t' \in \mathbb{T}, t' < t, \textit{ if } (q,g,t',q'',g'') \in T \\ \qquad \textit{then } \forall e' \in E_+, (q'',g'',e',q''',g''') \in T \Longrightarrow \\ \qquad\quad time \not<_0 e'. \\ \textit{otherwise if } (q,g,t,q'',g'') \in T, t \in \mathbb{T}, t \leq k, \\ \quad \textit{then } \forall e', (q'',g'',e',q''',g''') \in T \Longrightarrow e \not<_k e'. \end{cases}$$

We denote $\mathcal{A}\restriction\!< \;\; = \langle E_t, F_t, S_t, \pi, T\restriction\!<\rangle$.

Remark 3.3. Again, if $\mathbb{T} = \{0\}$, we obtain the definition of the priority relation restriction (Def. 2.3).

We can lift the following theorem for ITS stated in [10] to TITS:

Theorem 3.2. (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 $h : \mathcal{A}_1\restriction\!< \;\; \longrightarrow \mathcal{A}_2\restriction\!<$ is also a timed bisimulation homomorphism.

The proof is given in appendix A.2.

3.5. Timed Components

Timed AltaRica components are the timed extensions of AltaRica components (see Def. 2.4). 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 section 2.2.3.

Definition 3.7. (Timed Component)
A *timed component* is a tuple $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, < \rangle$ with:

1. V_S, V_F are finite sets for respectively *state* variables, *flow* variables with the property of being disjoint. We denote $V_T = V_S \cup V_F$; C_S, C_F are finite sets for respectively *clock* variables, *real 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. $E = E_+ \cup \{\varepsilon\}$ 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 *free*$(A) \subseteq V_T \cup C_T$; $A_{V_T} \in \mathbb{F}$, *free*$(A_{V_T}) \subseteq V_T$. $A_{C_T} = \bigwedge_{k \in K} P_k \implies I_k$ where K is a finite set of indices, $P_k \in \mathbb{F}$, *free*$(P_k) \subseteq V_T$, $I_k \in \mathbb{F}$, *free*$(I_k) \subseteq C_T$ and I_k defines a convex region of $\mathbb{R}^p$ if $|C_T| = p$;
4. $M \subseteq (\mathbb{F} \times \mathcal{B}(C_T)) \times E \times (\mathbb{E}(V_T)^{V_S} \times \mathcal{A}(C_T))$ is a *macro-transition* relation such that every $((g, \gamma), e, (a, R)) \in M$ satisfies:
   (a) (g, γ) is a *guard* such that $g \in \mathbb{F}$ and *free*$(g) \subseteq V_T$; $\gamma \in \mathcal{B}(C_T)$;
   (b) $e \in E$ is the *event* of the transition;
   (c) $a : V_S \to \mathbb{E}(V_T)$ is an *assignment* for the variables in V_S. $R \in \mathcal{A}(C_T)$ is the *clock assignment* of the transition;
5. $<$ is a timed priority relation.

Remark 3.4. Item 3 of Def. 3.7 allows us to specify constraints C between clock variables and real-valued flow variables (e.g. $Y = x$ where Y is a flow variable and x is clock variable): it suffices to use $tt \implies C$ where $C \in \mathbb{F}(C_T)$ (e.g. $tt \implies Y = x$).

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).

Example 3.1. (The Train)
In Fig. 4 the time features of component `TRAIN` appear on line 7 where a clock (state) variable `t` is declared; it is used to constrain the guards of the transitions (see lines 9 to 11) and on some of them `t` is reset; also the assertion (lines 16–17) implies that when in state *etat* $= 1$ (resp. 2) time cannot elapse after `t` has reached 30 (resp. 20).

Definition 3.8. (Semantics of Timed Components)
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 $|C_F| = m$. The *semantics* of $\mathcal{T}$ over the time domain $\mathbb{T}$ is the timed interfaced transition system $[\![\mathcal{T}]\!] = \langle E_t, F_t, S_t, \pi, T \rangle$ of dimension (n, m) constructed in the following way:

1. $E_t = E \cup \mathbb{T}$,
2. $F_t = \mathcal{D}^{V_F} \times \mathbb{R}^m$,

```
1:  node TRAIN
2:  flow  N : [0,1];
3:  event approach, in, exit;
4:  state
5:     etat : [0,2];
6:     n : [0,1];
7:     t : clock;      // Definition of a clock variable
8:  trans
9:   t >= 70 & etat=0 |- approach -> etat := 1, t := 0, n := 1;
10:  20 <= t <= 30 & etat=1 |- in -> etat := 2, t := 0;
11:  10 <= t <= 20 and etat=2 |- exit -> etat := 0, t := 0, n := 0;
12:  init
13:    etat=0,n=0,t=0;
14:  assert
15:    N=n;
16:   (etat=1) => (t<=30);      // Time assertions
17:   (etat=2) => (t<=20);
18:  edon
```

Figure 4. Specification of a Train as a Timed Component

3. $S_t = \{(s,\nu) \in \mathcal{D}^{V_S} \times \mathbb{R}^n_{\geq 0} | \exists (f,\mu) \in \mathcal{D}^{V_F} \times \mathbb{R}^m \,|\, ((s,\nu),(f,\mu)) \in [\![A]\!]\}$,
4. $\pi : S_t \to 2^{F_t}$ such that $\pi(q) = \{(f,\mu) |\ (q,(f,\mu)) \in [\![A]\!]\}$,
5. $T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ and $T = [\![M]\!] \!\!\upharpoonleft\!\!\succ$ with:
   (a) let $t = ((g,\gamma), e, (a,R))$, define $[\![t]\!]$ by:

$$((s,\nu),(f,\mu),e,(s',\nu'),(f',\mu')) \in [\![t]\!] \text{ if } \begin{cases} ((s,\nu),(f,\mu)) \in [\![A \wedge g \wedge \gamma]\!] \\ \wedge s' = a(s,f) \\ \wedge \quad \nu' = R(\nu,\mu) \\ \wedge \quad (f',\mu') \in \pi(s',\nu') \end{cases}$$

   with $R(\nu,\mu)$ the new clock assignment after resetting the variables in R.
   (b) let $\delta \in \mathbb{T}$, define $[\![\delta]\!]$ by:

$$((s,\nu),(f,\mu),\delta,(s,\nu'),(f,\mu')) \in [\![\delta]\!] \text{ if } \begin{cases} ((s,\nu),(f,\mu)) \in [\![A]\!] \\ \wedge \nu' = \nu + \delta \wedge ((s,\nu'),(f,\mu')) \in [\![A]\!] \\ \wedge \forall \delta' \leq \delta, \exists \mu_{\delta'} \,|\, (\nu + \delta', \mu_{\delta'}) \in [\![I(s,f)]\!] \end{cases}$$

   with $I(s,f) = \bigwedge_{k \in K | (s,f) \in P_k} I_k$.
   (c) $[\![M]\!] = \cup_{t \in M} [\![t]\!] \bigcup \cup_{\delta \in \mathbb{T}} [\![\delta]\!]$.

Remark 3.5. Note that $[\![I(s,f)]\!]$ is a convex set as it is a conjunction of convex sets. We have not used this property of $I(s,f)$ in the semantics of components as it is not required in this definition. Anyway in the sequel we will need this assumption and this is why we have put it in Def. 3.7 of timed components.

The delay transitions in the semantics of a timed components leave the "continuous" flows free to take any value as long as the invariant $I(s,f)$ is satisfied. This is rather permissive as the values

encountered along a delay transition could even be non continuous. For instance a constraint on a flow like $x \leq Y \leq x+2$ where x is a clock and Y a continuous flow would allow Y to take any value between x and $x+2$ at each time point. If we define flow to be clock we constrain the set of equations we can write in the assertion. Indeed equations like $Y = 2x$ could not be defined with a "clock" Y. So far we stick to this permissive definition and we will tackle later which kind of flows can be "implemented" (see section 4.4).

Finally in the case $C_F = \emptyset$ 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.)

As for the untimed case we have the following lemma:

Lemma 3.1. 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 priority relation. Then $[\![\langle V_S \cup C_S, V_F \cup C_F, E, A, M, < \rangle]\!] = [\![\langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset \rangle]\!] \restriction <$.

The proof is straightforward from Def. 3.6 and Def. 3.8.

3.6. Timed Nodes

Timed nodes are straightforward extensions of nodes. Indeed, if we assume as stated in Def. 2.6 of a node, that the synchronization constraint is expanded, the new constraint added by the time transitions is trivial: the synchronized time transitions for n nodes are of the form $(\delta, \delta, \cdots, \delta), \delta \in \mathbb{T}$ where $\mathbb{T}$ is the time domain and they do not need to be specified.

Definition 3.9. (Timed Node)
A *timed node* is a tuple $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\widetilde{V}, <_{\widetilde{V}}) \rangle$ with:

1. V_F is a set of *flow* variables,
2. C_F is the set of *real flow* variables,
3. $E = E_+ \cup \{\varepsilon\}$ 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; the interface of the node is composed of (i) $V_{F_i} \cup C_{F_i}$, the set of discrete flows and real flows of $\mathcal{N}_i$ and (ii) E_i the set of events of $\mathcal{N}_i$. 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 (discrete) flow variables of $\mathcal{N}_0$ is $V_{F_0} = V_F \cup V_{F_1} \cup V_{F_2} \cup \cdots \cup V_{F_n}$, and the set of real flow variables is $C_{F_0} = C_F \cup C_{F_1} \cup C_{F_2} \cup \cdots \cup C_{F_n}$,
7. $\widetilde{V} \subseteq E_0 \times E_1 \times \cdots \times E_n$ is an expanded synchronization set together with a priority relation $<_{\widetilde{V}}$. (see Def. 2.2).

Remark 3.6. Notice that $<_{\widetilde{V}}$ is a priority relation and not a timed priority relation. This is because $(\widetilde{V}, <_{\widetilde{V}})$ expresses the discrete synchronization constraint.

Example 3.2. (Hierarchical Specification of the Train-Gate-Controller)
A timed version of the train-gate-controller is given in Fig. 5. Notice that the gate is a component, the train is the component given by Fig. 1, but the node `MAIN` embeds the `CONTROLLER` node and plays the role of $\mathcal{N}_0$.

```
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 -> ;
  etat=1 & y <= 10 |- 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;
  etat=3 & y <= 10 |- up -> etat:=0;
 init
  etat:=0, y:=0;
assert
  (etat =1) => (y <= 10);
  (etat =3) => (y <= 10);
edon
```

(a) The Timed Gate

```
node MAIN
flow  N : [0,p];
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 -> ;
 etat=1 & z<=10 |- Go_down -> etat:=0;
 etat=2 & z <= 10 |- Go_up -> etat:=0;
 etat=2 |- approach -> etat:= 1, z:=0;
sub t1, t2 : TRAIN, g : GATE;
sync <t1.approach,t2.approach,approach>;
  <t1.approach,approach>;
  <t2.approach,approach>;
  <Go_down,g.Go_down>;
  <t1.exit,t2.exit,exit>;
  <t1.exit,exit>;
  <t2.exit,exit>;
  <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
```

(b) The Timed Controller

Figure 5. Timed AltaRica Specifications for the Controller and the Gate

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

Definition 3.10. (Semantics of Timed Nodes)
Let $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\widetilde{V}, <_{\widetilde{V}})\rangle$ be a timed node and $[\![\mathcal{N}_i]\!] = \langle E_{i_t}, F_{i_t}, S_{i_t}, \pi_i, T_i\rangle$ of dimension (n_i, m_i) for $i \in [0, n]$. The semantics of $\mathcal{N}$ is the timed interfaced transition system $[\![\mathcal{N}]\!] = \langle E_t, F_t, S_t, \pi, T\rangle$ of dimension $(\sum_{k=0}^{n} n_i, |C_F|)$ defined by:

1. $E_t = E \cup \mathbb{T}$,
2. $F_t = \mathcal{D}^{V_F} \times \mathbb{R}^m$, with $m = |C_F|$,
3. for $q_i \in S_{i_t}$, let $\overline{q} = (q_0, q_1, \cdots, q_n)$, then

$$\pi(\overline{q}) = \{(f,\mu) \in \mathcal{D}^{V_F} \times \mathbb{R}^m \,|\, \forall i \in [1,n], \exists \eta_i \in \pi_i(q_i) \,|\, ((f,\mu), \eta_1, \eta_2, \cdots, \eta_n) \in \pi_0(q_0)\}$$

4. $S_t = \{q \in S_{0_t} \times S_{1_t} \times \cdots \times S_{n_t} \,|\, \pi(q) \neq \emptyset\}$;
5. $T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ is defined by:
   (a) let $<_0$ be the timed priority relation defined by:

$$(e_0, e_1, \cdots, e_n) <_0 (e'_0, e'_1, \cdots, e'_n) \Longleftrightarrow e_0 < e'_0$$

   (b) let $e = (e_0, e_1, \cdots, e_n) \in E_0 \times \cdots \times E_n \cup \{(\delta, \cdots, \delta)\}$, $\overline{s} = (s_0, s_1, \cdots, s_n)$ and $\overline{s}' = (s'_0, s'_1, \cdots, s'_n)$. Define $T_{\mathcal{N}}$ by:

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

   (c) then $T = \left(T_{\mathcal{N}} |\!<_{\tilde{V}}\right) |\!<_0$.

We have the node version of lemma 3.1:

Lemma 3.2. Let $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\widetilde{V}, <_{\widetilde{V}}) \rangle$ be a timed node and $[\![\mathcal{N}]\!]$ its semantics. Then $[\![\mathcal{N}]\!] = [\![\langle V_F, C_F, E, \emptyset, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\widetilde{V}, <_{\widetilde{V}}) \rangle]\!] |\!<$.

The proof is straightforward from Def. 3.9 and Def. 3.10.

The semantics of nodes is compositional with respect to timed bisimulation:

Theorem 3.3. Let $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\widetilde{V}, <_{\widetilde{V}}) \rangle$ and $\mathcal{N}' = \langle V_F, C_F, E, <, \mathcal{N}'_0, \cdots, \mathcal{N}'_n, (\widetilde{V}, <_{\widetilde{V}}) \rangle$ be two timed nodes such that $\forall i \in [0..n]$ there is a timed homomorphism h_i from $[\![\mathcal{N}_i]\!]$ to $[\![\mathcal{N}'_i]\!]$. Then there exists a timed homomorphism h from $[\![\mathcal{N}]\!]$ to $[\![\mathcal{N}']\!]$.

The proof is given in appendix A.3.

Timed AltaRica is a hierarchical modeling language so that each timed node can be expressed by a timed component. The timed priorities and the synchronization are directly encoded into the resulting timed component. Let $\mathcal{N} = \langle V_F \cup C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}}) \rangle$ be a timed node, we present the construction (extending the one given in [10]) 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.

First we associate to each timed node a timed component defined as follows:

Definition 3.11. (Symbolic Semantics)
If $\mathcal{N} = \langle V_F \cup C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$ is a timed node, with $\mathcal{N}_i = \langle V_{F_i} \cup C_{F_i}, E_i, <_i, \mathcal{N}_{i_0}, \cdots, \mathcal{N}_{i_n}, (\tilde{V}_i, <_{\tilde{V}_i})\rangle$ for $0 \leq i \leq n$, we denote by $\mathcal{C}_\mathcal{N} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset\rangle$ the timed component constructed as follows:

1. $\forall 0 \leq i \leq n$
   (a) if $\mathcal{N}_i$ is a timed component, then we define $\mathcal{N}'_i = \mathcal{N}_i{\restriction}<_i$ and the timed priority is syntactically encoded in $\mathcal{N}'_i$ as defined later in section 3.7.2;
   (b) if $\mathcal{N}_i$ is a timed node, then we define $\mathcal{N}'_i = \mathcal{C}_{\mathcal{N}_i}$ the rewriting of $\mathcal{N}_i$ into a timed component and encode timed priority syntactically as defined in section 3.7.2;
   (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. $V_S = V'_{S_0} \cup \cdots \cup V'_{S_n}$ and $C_S = C'_{S_0} \cup \cdots \cup C'_{S_n}$;
3. $A = (\exists_{i=1..n}(V'_{F_i} \cup C'_{F_i}))\,.\,\bigwedge_{i=0..n} A'_i$;
   where the notation $\exists_{i=1..n}(W_i).\phi$ stands for: $\forall i, \exists \eta_i \in W_i$ such that $\phi(\eta_i)$. For $((s,\nu),(f,\mu)) \in \mathcal{D}^{V_S} \times \mathbb{R}^{C_S} \times \mathcal{D}^{V_F} \times \mathbb{R}^{C_F}$ we define:
   - $((s,\nu),(f,\mu)) \in [\![A]\!] \iff \forall i \in [1..n], \exists \eta_i \in \mathcal{D}^{V'_{F_i}} \times \mathbb{R}^{C'_{F_i}}$ *s.t.* $((s,\nu),(f,\mu),\eta_1,\cdots,\eta_n)) \in [\![\bigwedge_{i=0..n} A'_i]\!]$,
   - $\forall i \in [1..n], ((s,\nu),(f,\mu),\eta_1,\cdots,\eta_n)) \in [\![A'_i]\!] \iff ((s_i,\nu_i),\eta_i) \in [\![A'_i]\!]$.
4. the set of macro-transitions $M \subseteq (\mathbb{F} \times \mathcal{B}(C_T)) \times E \times (\mathbb{E}(V_T)^{V_S} \times \mathcal{A}(C_T))$ is defined by $M = (M' \restriction <_{\widetilde{V}}){\restriction}<_0$, where $<_0$ is the timed priority relation specified in Def. 3.10, and $((g,\gamma),e,(a,R)) \in M'$ if and only if:
   - $\forall 0 \leq i \leq n$, there is a transition $((g_i,\gamma_i),e_i,(a_i,R_i)) \in M'_i$ such that:
     - $g = (\exists_{i=1..n} V_{F_i}).g_0 \wedge \cdots \wedge g_n$,
     - $\gamma = (\exists_{i=1..n} C_{F_i})\,.\gamma_0 \wedge \cdots \wedge \gamma_n$,
   - $\forall x \in V_S \cap V'_{S_i}$ we have $a(x) = a_i(x)$ and $\forall c \in C_S \cap C'_{S_i}$ we have $R(c) = R_i(c)$,
   - $e = (e_0, e_1, \cdots, e_n) \in \widetilde{V}$.

Theorem 2.1 (see page 1009) for untimed nodes carries over to timed nodes:

Theorem 3.4. Let $\mathcal{N}$ be a timed node. Then $\mathcal{N}$ can be rewritten into a timed component $\mathcal{C}_\mathcal{N}$ such that $[\![\mathcal{N}]\!]$ and $[\![\mathcal{C}_\mathcal{N}]\!]$ are timed bisimilar.

The proof is given in appendix A.4.

3.7. Syntactical Timed Priority

In [27] 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 $<$ be a timed priority relation. We first assume[6] that $<$ contains no urgent events *i.e.* $\forall e \in E, \mathit{time} \not< e$. Our aim is to compute the transition relation $M|<$ syntactically *i.e.* by finding new guards that define $M|<$. 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 (section 3.7.1) into the existing guards. Then we show how to encode timed priority (section 3.7.2) again by strengthening the guards. Finally we detail how urgency is handled in section 3.7.3.

3.7.1. 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) $\pi(a(s,f),R(\nu,\mu)) \neq \emptyset$, *i.e.* there are some admissible flow values. 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. 2.4). Let $t = (g,e,a)$ be a transition of this component, and A the assertion. For $Q \subseteq S \times F$, we define $\mathit{Pre}_t(Q) = \{(s,f) \,|\, \exists f' \,|\, (a(s,f),f') \in Q\}$. Assume $\mathit{Pre}_t(\llbracket A \rrbracket)$ can be defined by a formula $\phi_t \in \mathbb{F}$, and $\mathit{free}(\phi_t) \subseteq V_T$. 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 $\mathit{Pre}_t(Q) = \{((s',\nu'),(f',\mu')) \,|\, \exists f'',\mu'' \,|\, ((a(s',f'),R(\nu',\mu')),(f'',\mu'')) \in Q\}$. Assume $\mathit{Pre}_t(\llbracket A \rrbracket)$ can be written as $\phi_t \wedge \theta_t$ with $\mathit{free}(\phi_t) \subseteq V_T$ and $\mathit{free}(\theta_t) \subseteq C_T$.

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 $\mathit{Pre}_t(\llbracket A \rrbracket)$ into guards of the form (g,γ) with $g \in \mathbb{F}, \mathit{free}(g) \subseteq V_T$ and $\gamma \in \mathcal{B}(C_T)$. Assume $A = p_1 \wedge p_2 \wedge \cdots \wedge p_n \wedge (q_1 \implies i_1) \wedge \cdots \wedge (q_l \implies i_l)$. Assume $\mathit{Pre}_t(\llbracket A \rrbracket)$ is a conjunction of the form[7] $p'_1 \wedge p'_2 \wedge \cdots \wedge p'_k \wedge (q'_1 \implies i'_1) \wedge \cdots \wedge (q'_m \implies i'_m)$. Let $P' = p'_1 \wedge p'_2 \wedge \cdots \wedge p'_k$. We can rewrite $\mathit{Pre}_t(\llbracket A \rrbracket)$ as[8]:

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

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

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

[6] Urgency is dealt with in section. 3.7.3 and requires additional assumptions and definitions.

[7] Quantifier elimination in $\mathit{Pre}_t(\llbracket A \rrbracket)$ can only be done under some conditions (e.g. the discrete domain is finite). We do not discuss this in this paper and assume we can actually find a quantifier-free expression for $\mathit{Pre}_t(\llbracket A \rrbracket)$.

[8] This expression is equivalent to the one given in [10], Def. 9.2.3 (Priorités Syntaxiques), page 85.

It remains to replace t by the $t_p, p \in [1..s]$ to build a new timed component and t can be fired in the original component if and only if one of the t_p can be fired in the new component (leading to the same values for the state variables.) In the sequel we assume guards have been strengthened so that if a guard evaluates to true then the transition can actually be fired.

3.7.2. Encoding Timed Priority

The Simple Case. Let $t = ((g, \gamma), e, (a, R)) \in M$. Assume $e <_k e'$ and there is only one transition $t' = ((g', \gamma'), e', (a', R')) \in M$ labelled with e'. Then t can be fired from a configuration q only if (g, γ) is true in q and:

1. either g' is not true in q,
2. or g' is true in q and γ' will not be true within k time units.

First we deal with the discrete part of the guard and split the transition t into t_1 and t_2 with:

- $t_1 = ((g \wedge \neg g', \gamma), e, (a, R))$ which corresponds to item 1 above;
- $t_2 = ((g \wedge g', \gamma), e, (a, R))$ which corresponds to item 2 above although γ needs to be strengthened to meet the requirements of item 2 above.

We now show how to strengthen γ in t_2. A useful operator was introduced in [28, 29] for this purpose:

Definition 3.12. (Modal Operator [28, 29])
Let $X = \{x_1, x_2, \cdots, x_n\}$. Let $\nu \in \mathbb{R}^n$ and $k \in \mathbb{N}$. Let $\phi \in \mathcal{B}(X)$ and $\mathbb{T}$ be the time domain. We define the (state) predicate $\Diamond_k \phi$ by:

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

Now we strengthen the guard γ in t_2 and define $t_2' = ((g \wedge g', \gamma \wedge (\neg \Diamond_k \gamma')), e, (a, R))$. According to [28, 29], it is possible to eliminate the existential quantifier in $\Diamond_k \gamma'$ and to obtain a quantifier-free formula (we will not get into the details and refer the reader to [28, 29]).

General Case. In the general case there could be p transitions $t_i' = ((g_i', \gamma_i'), e_i, (a_i', R_i'))$ s.t. $e <_{k_1} e_1, \cdots, e <_{k_p} e_p$. Then we split t into 2^p transitions $t_F = ((g_F, \gamma_F), e, (a, R))$ with $F \subseteq [1..p]$:

$$g_F = g \wedge \bigwedge_{i \in [1..p] \setminus F} \neg g_i' \wedge \bigwedge_{i \in F} g_i' \tag{2}$$

$$\gamma_F = \gamma \wedge \bigwedge_{i \in F} \neg \Diamond_{k_i} \gamma_i' \tag{3}$$

Remark 3.7. As stated in remark 3.2, page 1014, we do not modify the *invariants* of the system.

According to [27], the formula $\Diamond_k \gamma$ can be written as simple formula in $\mathcal{B}(C_T)$. If we denote by $M \!\mid\!<$ the transition relation obtained by:

1. strengthening the guards by the weakest precondition for *fireability* as defined in section 3.7.1,

2. strengthening the guards to encode the timed priority relation as defined above in this subsection,

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

Lemma 3.3. (Syntactical Priority)
$[\![\mathcal{T}|\!<]\!] = [\![\mathcal{T}]\!]|\!<$.

Lemma 3.3 follows from Def. 3.10 and Def. 3.12. From lemma 3.1, we obtain the following corollary:

Corollary 3.1. $[\![\langle V_S \cup C_S, V_F \cup C_F, E, A, M, <\rangle]\!] = [\![\langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset\rangle|\!<]\!]$

This completes the syntactical encoding of timed priority without urgency for timed components.

3.7.3. Encoding Urgency

Urgency consists in preventing time elapsing when a discrete transition is enabled. In this section we assume the time domain is $\mathbb{R}_{\geq 0}$. Also we assume time determinism and denote $\nu - t$ the valuation defined by $(\nu - t)(x) = \nu(x) - t$ (time non determinism is more technically involved but can be handled as well). Our work is based on previous papers by S. Bornot, J. Sifakis and S. Tripakis [28, 29, 27]. The authors define the notion of *rising edge* of a guard that plays a central role:

Definition 3.13. (Rising Edge [27])
Let $X = \{x_1, x_2, \cdots, x_n\}$, $\nu \in \mathbb{R}^n$ and $\gamma \in \mathcal{B}(X)$. The *rising edge* of γ, denoted $\gamma\uparrow$ is the predicate defined by:

$$\gamma\uparrow(v) = \big(\gamma(v) \wedge \exists t > 0\,,\, \forall 0 < t' \leq t,\ \neg\gamma(v - t')\big) \vee \big(\neg\gamma(v) \wedge \exists t > 0, \forall 0 < t' \leq t,\ \gamma(v + t')\big) \quad (4)$$

We assume that each guard of a transition labelled by an urgent event is such that $[\![\gamma\uparrow]\!] \subseteq [\![\gamma]\!]$. Indeed $x > 10$ is not a relevant guard for an urgent transition as there is no *first instant* at which the guard becomes true: the transition becomes urgent strictly after 10 which is a fuzzy instant and this is in contradiction with urgency. Note that in this case $(x > 10)\uparrow \equiv (x = 10)$ which gives the same rising edge as for $x \geq 10$ but the latter has a first instant for which it is true. This problem is well-known and is already discussed in [29]. Note that in this case equation (4) of Def. 3.13 simplifies in $\gamma\uparrow(v) = \big(\gamma(v) \wedge \exists t > 0\,,\, \forall 0 < t' \leq t,\ \neg\gamma(v - t')\big)$. We also assume that a guard γ of an urgent transition is convex and this implies that $\gamma\uparrow$ is convex as well.

Urgency as an assertion The semantics of urgency (see Def. 3.6) implies that when a transition becomes urgent (*i.e.* its guard is true) time elapsing is forbidden and this is the semantics proposed in [29]. It does not imply that the urgent transition is fired. Also this notion is different from the notion of urgency in UPPAAL [20] which only constrains processes to synchronize on common channels (synchronized events) whenever they can.

To be more precise assume we have a component `U` with an urgent transition e_u as defined on Fig. 6(a).

Start in configuration $(s = 0, x = 0)$. At some point f can occur. If it occurs before $x = 10$ it is possible to let some time elapse until $x = 10$ reaching $(s = 1, x = 10)$. At this point the urgent transition prevents time from elapsing. Anyway a new occurrence of f could occur and set s to 0 again: in this

```
1:  node U
2:    state
3:      s : [0,1];
4:      x : clock;
5:    event
6:      e_u > time, f
7:    trans
8:      s=1 & 10<= x<=20 |- e_u -> s:=0;
9:      |- f -> s:=0;
10:     |- f -> s:=1;
11:   init s:=0, x:=0 ;
12: edon
```

(a) Node U with event e_u urgent

```
1:  node U_Y
2:    flow Y ;    // the flow variable Y
3:    state
4:      s : [0,1];
5:      x : clock;
6:    event
7:      e_u, f
8:    trans
9:      s=1 & 10<=x<=20 |- e_u -> s:=0;
10:     |- f -> s:=0;
11:     |- f -> s:=1;
12:   init s:=0, x=0 ;
13:   assert
14:     (s=1) => (10<x<=20 => Y=0)
15: edon
```

(b) Urgency as an assertion

Figure 6. Encoding Urgency

configuration the urgent event e_u is no more enabled and time can elapse. Thus to use our notion of urgency to force a transition to occur, one must ensure that once an urgent transition is enabled ($x = 10$) no other transition can disable it (to achieve this, one could change the enabling condition `True` of line ℓ_9 of node U to `x<10`).

To encode urgency, we use an additional real flow variable Y (see line 2 of Fig. 6(b)). This flow variable is assumed to be reset to 0 on each discrete transition and evolves at rate 1 (synchronous with physical time) on delay transitions. How this will be achieved will be dealt with later in this section. The syntactical encoding of urgency consists in adding a timed invariant (line 14 of node `U_Y`, Fig. 6(b)) to constrain time elapsing. Note that this assertion implies $Y = 0$ only when $x > 10$ (and not $x \geq 10$). Intuitively, assume we reach a configuration $(s = 1, x < 10)$. Then time can elapse from this configuration until $x = 10$. Indeed $(s = 1, x = 10)$ satisfies Def. 3.6 as for each strictly preceding instant the assertion is true. From this configuration on time cannot elapse as $Y > 0$ and the assertion forbids it. Now if we reach $(s = 1, 10 \leq x \leq 20)$ by firing a discrete transition, Y is set to 0 and time elapsing is also forbidden. This achieves urgency (in the sense that time elapsing is prevented).

Some limitations of our encoding is that we do not know how to deal with urgent transitions with *sharp* urgent guards as $x = 5$. This is why we require an additional assumption on guards for urgent transitions: the (temporal) guard γ_u must satisfy $\exists \epsilon > 0\,,\ \nu \in [\![\gamma_u{\uparrow}]\!] \implies \forall \epsilon' \leq \epsilon \quad \nu + \epsilon' \in [\![\gamma_u]\!]$. We refer to this latter property as γ_u is not sharp.

Correctness of the encoding Let $\mathcal{T} = (V_S \cup C_S, V_F \cup C_F, E, A, M, <)$ be a timed component where $<$ consists in one element: $e_u > \textit{time}$ (e_u is urgent). Assume there is one urgent transition $t_u = ((g_u, \gamma_u), e_u, (a_u, R_u))$, γ_u is not sharp, and there is a flow variable Y that is reset on each discrete transition and evolving at rate 1 on delay transitions.

Define the timed component $\mathcal{T}_u = (V_S \cup C_S, V_F \cup C_F, E, A \wedge \varphi_u, M, \emptyset)$ with $\varphi_u \stackrel{def}{=} g_u \implies \big((\gamma_u \wedge \neg(\gamma_u{\uparrow})) \implies Y = 0\big)$. Note that we assume Y is an *invisible variable* that does not belong to C_F. This is just for the sake of clarity as otherwise we need to define a new notion of timed bisimilarity

for timed interfaced transition systems that do not have the same sets of flow variables (remind that Def. 3.2 imposes the two systems to have the same interface).

Theorem 3.5. $[\![\mathcal{T}]\!]$ and $[\![\mathcal{T}_u]\!]$ are timed bisimilar.

The proof is given in appendix A.5.

Implementation of the encoding To implement our encoding and add a fresh flow variable Y, we proceed as follows:

1. create node `U_Y` from node `U`, as described by Fig. 6(b),
2. build a new node `YY` (Fig. 7(a)) that manages a variable Y that satisfies the assumptions we needed before: Y is reset on each discrete transition and evolves at rate 1 on delay transitions. Each discrete event of other components will be synchronized with event u of `YY`;
3. build a parent node `UU` that synchronizes `U_Y` and `YY`; this node is given in Fig. 7(b).

```
node YY
 flow   Y;
 event  u;
 state
    y : clock;
 trans
   |- u -> y:=0 ;
 init   y:=0;
 assert Y=y
edon
```

(a) Node `YY`

```
node UU
 event  e_u,f;
 sub    CU_Y:U_Y; C_YY:YY;
 sync
    <f,C_YY.u,CU_Y.f>;
    <e_u,C_YY.u,CU_Y.e_u>;
 assert
    CU_Y.Y=C_YY.Y
edon
```

(b) Node `UU`

Figure 7. Hierarchical Modeling of Urgency

This scheme can be carried out for multiple urgent events. We do not detail this in this paper as it is just a technical exercise.

Now that we know how to encode timed priority syntactically and how to flatten a node into a component. We proceed with a translation of timed components into timed automata. This will enable us to check various timed properties.

4. From Timed Nodes to Timed Automata

In this section, we present a translation of Timed AltaRica specifications to timed automata [17]. 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 [5], CMC [6] or KRONOS [31]. Notice that thanks to theorem 3.4 we only need to define the translation for timed components.

4.1. From Timed AltaRica Components to Timed Automata

Let $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset \rangle$ be a timed component with $|C_S| = n$ and $|C_F| = m$ (thanks to lemma 3.3, we can assume the timed priority relation of $\mathcal{T}$ is the empty relation). From Def. 3.7, 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 *time invariant* to a predicate on the discrete variables.

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

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 define the translation of a timed component with no flow variables and explain later how we deal with components with flow variables.

We write $G \uplus L = [1..p]$ as a shorthand for $G, L \subseteq [1..p], G \cap L = \emptyset, G \cup L = [1..p]$ *i.e.* G and L form a partition of $[1..p]$.

Definition 4.1. (Timed Automaton Associated with a Timed Component)
Let $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset \rangle$ be a timed component with $V_F = C_F = \emptyset$. Let $A = A_{V_T} \wedge A_{C_T}$ with $\mathit{free}(A_{V_T}) \subseteq V_T$, $A_{C_T} = \bigwedge_{j=1}^{p}(P_j \implies I_j)$ and $\mathit{free}(P_j) \subseteq V_T$ and $\mathit{free}(I_j) \subseteq C_T$. Given $G \subseteq [1..p], L \subseteq [1..p]$, we define:

$$r_G^L = \left(\wedge_{j \in G} P_j\right) \wedge \left(\wedge_{j \in L} \neg P_j\right) \quad (5)$$

$$l_G^L = A_{V_T} \wedge r_G^L \quad (6)$$

The timed automaton[9] $A(\mathcal{T}) = (L, L_0, E, X, I, T)$ associated with $\mathcal{T}$ is defined by:

- $L = \{l_G^L \mid G \uplus L = [1..p] \wedge [\![l_G^L]\!] \neq \emptyset\}$ is the set of locations[10],
- $L_0 = L$ is the set of initial states (actually in real Timed AltaRica specifications, a set of initial states is given as in the example of Fig. 5(b); assume this set is defined by a predicate *init* then $L_0 = init$),
- E is the set of events,
- $X = C_T$ is the set of clocks,
- the invariant I is defined by: $I(l_G^L) = \begin{cases} \wedge_{k \in G} I_k \text{ if } G \neq \emptyset \\ tt \text{ otherwise} \end{cases}$
- the transition relation T is defined by: let $l_G^L, l_{G'}^{L'} \in L$ such that $[\![l_G^L \wedge g]\!] \neq \emptyset$ and $a([\![l_G^L \wedge g]\!]) \cap [\![l_{G'}^{L'}]\!] \neq \emptyset$ (the source location intersects the discrete guard and the target location intersects the discrete part of the state space) and $t = ((g, \gamma), e, (a, R)) \in M$, then

$$(l_G^L, (g \wedge \mathit{Pre}_t(l_{G'}^{L'}) \wedge \gamma, e, (a, R)), l_{G'}^{L'}) \in T$$

[9] see section 3.1 for the definition of a timed automaton

[10] Thus the number of locations 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} and that $[\![l_G^L]\!] \subseteq S_t \times F_t$.

Remark 4.1. As we have imposed that the I_k denote convex sets, the invariants $I(l_G^L)$ are allowed by the definition of timed automaton (section 3.1).

In the previous definition we assume we can give constraints on the discrete variables in the guards, and allow assignments of the discrete variables on a transition, which is not formally allowed by the definition of timed automata of section 3.1, but this definition can trivially be extended to include this (timed automata of UPPAAL [5] allow the use of such features). Also if we do not make any assumption on the domain of the discrete variables of a Timed AltaRica specification the number of locations may be infinite. Anyway we can define the translation of a timed component into a timed automaton (with potentially an infinite number of locations):

Theorem 4.1. Let $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset \rangle$ be a timed component with $V_F = C_F = \emptyset$. Then $[\![\mathcal{T}]\!]$ and $[\![A(\mathcal{T})]\!]$ are timed bisimilar.

The proof is given in appendix A.6.

4.2. The Train Example

We now apply the previous translation to the train example of Fig. 8.

```
1:  node TRAIN
2:  // flow N : [0,1]; commented out
3:  event approach, in, exit;
4:  state
5:     N : [0,1];      // N is now a state variable
6:     etat : [0,2];
7:     n : [0,1];
8:     t : clock;
9:  trans
10:  t >= 70 & etat=0 |- approach -> etat := 1, t := 0, n := 1, N:=1;
11:  20 <= t <= 30 & etat=1 |- in -> etat := 2, t := 0;
12:  10 <= t <= 20 and etat=2 |- exit -> etat := 0, t := 0, n := 0, N:=0;
13: init
14:   etat:=0;n:=0;N:=0;t:=0;
15: assert
16:       // N=n; commented out
17:   (etat=1) => (t<=30);
18:   (etat=2) => (t<=20);
19: edon
```

Figure 8. Train Timed Component with no Flow Variables

The flow variable `N` is first assumed to be a state variable (line 5) and an assignment is given for `N` on lines 10–12. We assume that this enables us to get rid of the assertion on N (*i.e.* line 16 is commented out). Table 1 gives the locations and invariants of the corresponding timed automaton.

The next step consists in computing the graph structure: the result for the train component of Fig. 8 is given on Fig. 9. Notice that we compute an abstract timed automaton in the sense that discrete variables are not interpreted: as a result the number of locations of the timed automaton where the discrete variables are interpreted might be larger than the number of locations of this abstract timed automaton, even could be infinite.

$G \uplus L$	r_G^L	l_G^L	$I(l_G^L)$
$\emptyset \uplus \{1,2\}$	*etat* $= 0$	*etat* $= 0$	tt
$\{1\} \uplus \{2\}$	*etat* $= 1$	*etat* $= 1$	$t \leq 30$
$\{2\} \uplus \{1\}$	*etat* $= 2$	*etat* $= 2$	$t \leq 20$
$\{1,2\} \uplus \emptyset$	ff	ff	–

Table 1. Locations and Invariants of the Train Component

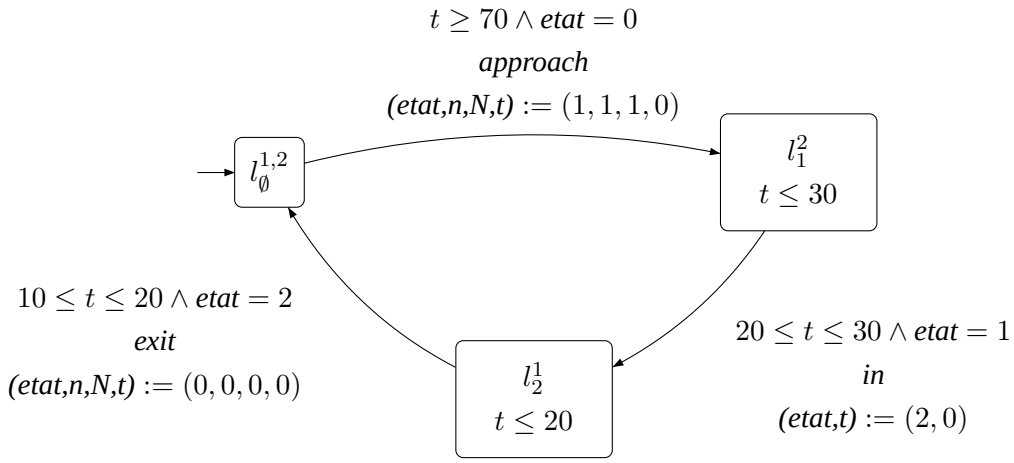


Figure 9. Translation of the Train component

4.3. Discussion of our Translation

The assumption that flow variables become clock state variables of the timed component means that the flows evolution rates and resets must follow the evolution rules of a clock in a timed automaton. This imposes restriction on the type of equations one can write in the assert part of a Timed AltaRica program. We will not go into details about it and the reader is referred to [32] for an exhaustive presentation. Nevertheless such constraints like $Y = x$, $Y = x + 1$ etc. (where Y is a flow and x a clock) can be dealt with in the translation into timed automata. More complex equations $Y = 2x + y$ can be handled using *hybrid automata* [18]. Constraints like $x + 1 \leq Y \leq 2y + x$ cannot be encoded into linear hybrid automata as the slope of Y is unbounded. In the sequel we assume that only assertions of the simple type $Y = x$ or $Y = x + c$, $c \in \mathbb{N}$ are used so that we can encode them into timed automata constraints. Computing the assignments of the flow variables that have become state variables is in this case easy and detailed in Table 3.

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

- we do not want to have an expensive computation to produce the timed automaton; 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 only perform syntactical rewriting;
- we do not want to constrain the discrete variables to be in a finite domain before doing 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 these 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.

4.4. Reachability Issues for Timed AltaRica Components

Timed AltaRica components are translated into timed automata as described previously. If the domain of the discrete variables is finite, we obtain a timed automaton with a finite number of discrete states. Reachability is decidable for timed component if the translation of a timed component belongs to a class of timed automata for which reachability is decidable. This problem has been extensively studied and an exhaustive set of results was given in [24]. We recall (see section 3.1) that the set of *clock constraints* $\mathcal{B}(X)$ over a set X of clocks is defined inductively by the grammar (see equation 1):

$$g := x \backsim r |\ x - y \backsim r\ | g \wedge g\ | g \vee g$$

with $x, y \in X, \backsim \in \{<, <, >, \geq, =\},\ r \in \mathbb{Z}$. The set of *diagonal-free* constraints $\mathcal{B}_{df}(X)$ is defined by the sub-grammar:

$$g := x \backsim r | g \wedge g\ | g \vee g$$

with $x \in X, \backsim \in \{<, <, >, \geq, =\}$, $r \in \mathbb{N}$. Table 2 gives a summary of the results in [24] (an assignment of the form $x :< c$ means that x is assigned any value less than c) concerning the decidability of the reachability problem for timed automata: decidability depends on the type of guards of the automata and on the type of assignments allowed.

Deterministic Assignment		Guards in $\mathcal{B}_{df}(X)$	Guards in $\mathcal{B}(X)$
$x := c$	(1)	Decidable	Decidable
$x := y$	(2)		
$x := x + 1$	(3)		Undecidable
$x := y + c$	(4)		
$x := x - 1$	(5)	Undecidable	
Non-Deterministic Assignment		**Guards in $\mathcal{B}_{df}(X)$**	**Guards in $\mathcal{B}(X)$**
$x :< c$	(6)	Decidable	Decidable
$x :> c$	(7)		Undecidable
$x :\backsim y + c$	(8)		
$y + c <: x :< y + d$	(9)		
$y + c <: x :< z + d$	(10)	Undecidable	

Table 2. Decidability Results for Reachability in Timed Automata (from [24])

In our setting, the decidability of reachability depends on the type of guards and assignments of the timed component as well as on the type of assertions used to constrain the continuous flow variables. If we allow only assertions on the continuous flow variables of the form $Y = x + c'$ where Y is a continuous flow variable, x is a clock state variable and $c' \in \mathbb{N}$ then the updating of Y on discrete transitions can be encoded as a clock assignment according to the encoding described in Table 3: in the case of non-deterministic assignments (6–10) of x we encode the assignment for Y with an ϵ-transition that occurs right after the one assigning x without any time elapsing (can be implemented by *committed* locations in UPPAAL for instance).

Combining Tables 2 and 3 we obtain that for an assertion containing only constraints of the form $Y = x + c'$ with Y a flow variable and x is a clock variable, $c' \in \mathbb{N}$:

- if all the guards on the clock and continuous flow variables are in $\mathcal{B}_{df}(X)$, reachability is decidable in case the assignments of the clock state variables are not of type (5) nor (10);
- if all the guards on the clock and continuous flow variables are in $\mathcal{B}(X)$, reachability is decidable only if the assignment of the clock state variables are of the form (1) or (2) and $c' = 0$ ($Y = x$ and $x := y$ allowed).

5. A Case Study Using Timed Priorities

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

Type of Assignment for x		Type of Assignment for Y	
$x := c$	(1)	$Y := c + c'$	(1)
$x := y$	(2)	$Y := y + c'$	(4)
$x := x + 1$	(3)	$Y := Y + 1$	(3)
$x := y + c$	(4)	$Y := y + c + c'$	(4)
$x := x - 1$	(5)	$Y := Y - 1$	(5)
$x :< c$	(6)		
$x :> c$	(7)		
$x :\backsim y + c$	(8)	ϵ-trans. $Y := x + c'$	(4)
$y + c <: x :< y + d$	(9)		
$y + c <: x :< z + d$	(10)		

Table 3. Encoding Flow Variable Y constrained by $Y = x + c'$

- there are two tracks crossing at the gate,
- the trains can come from any side on these two tracks,
- the aim is to ensure property P stating "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 [5] (note in this case we do have to instantiate k with a value in $\mathbb{N}$ before using the tool).

5.1. Translation of the Train-Gate-Controller into Timed Automata

The components Train-Gate-Controller have been given in Fig. 1, page 1003 and Fig. 5, page 1018.

From those components we can build timed automata using the algorithm defined in section 4.1. For this particular case of a hierarchical node with sub-components we can use an alternative way for building the timed automaton $A(Main)$: it is the synchronized product of the three automata obtained by translating each component into a timed automaton. The timed automata for nodes `TRAIN` and `GATE` are given in Fig. 10. The timed automaton[11] corresponding to node `MAIN` is given in Fig. 11. `MAIN.N`.

On the synchronized product of the three UPPAAL timed automata, property P is given by the UPPAAL-style property:

```
A[]((TRAIN1.s2 or TRAIN2.s2) imply GATE.etat==2)
```

We can check that P is satisfied for any fixed value of k.

[11] To deal with priority, we use the algorithms given in section 3.7 and we obtain a priority free component.

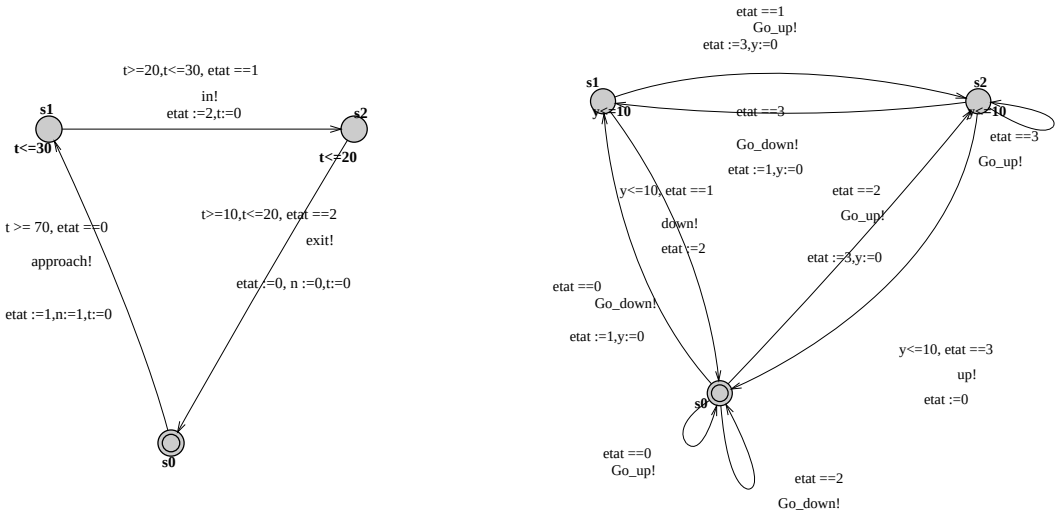


Figure 10. Train and Gate Automata in UPPAAL

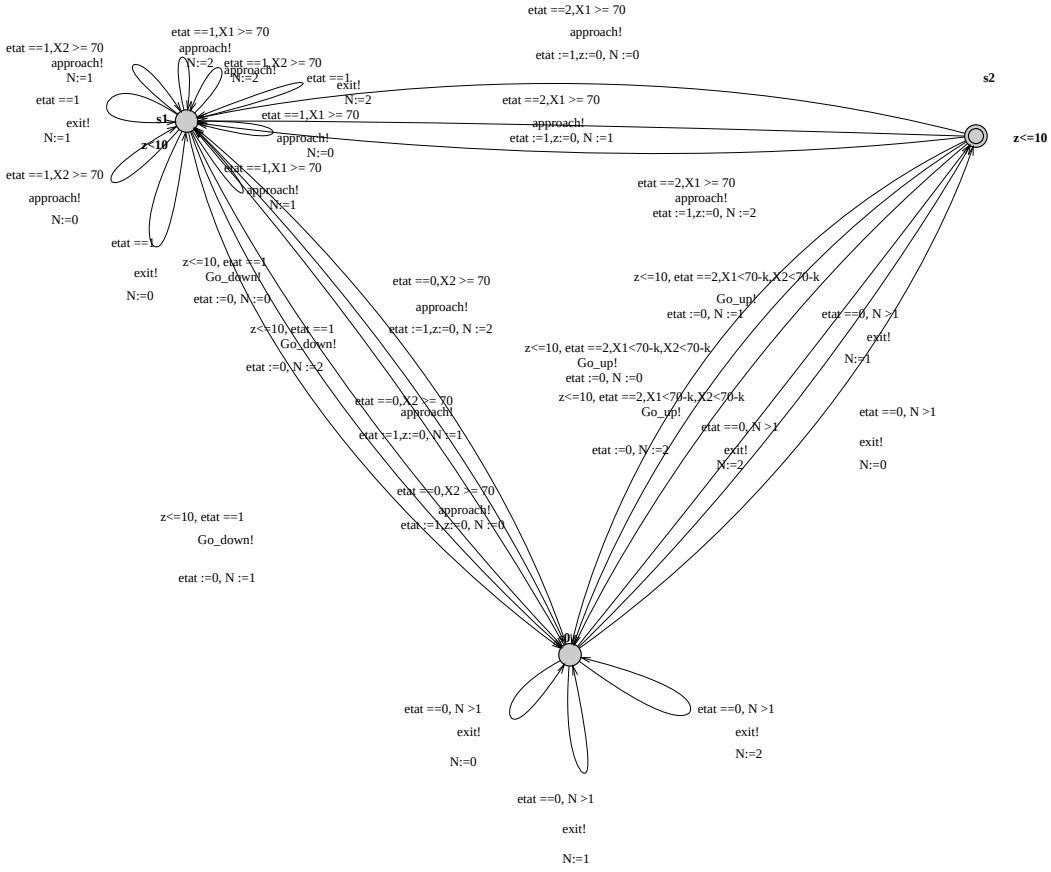


Figure 11. Controller Automata in UPPAAL

5.2. Influence of Timed Priorities

Timed priorities constrain the system and one of the first questions that arises is what kind of behaviour do we forbid. The priority we have given in Fig. 5(b), page 1018 means that we do not want to raise the gate if a new train can enter the near section within less than k time units. As intended, there should be a threshold value k_0 for k such that for all $k \geq k_0$, the gate remains closed forever. We can express this as the property[12] `A[](GATE.etat==2)` and it is satisfied for $k_0 = 40$.

Another interesting problem concerns the liveness of the system. As stated in remark 3.2, page 1014, some deadlock may occur when prioritising the system. The train-gate-controller without priority is deadlock free and looses this property as soon as $k > 0$.

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 3.2),
- a timed component with timed priorities can syntactically be rewritten into a timed component without timed priorities and has the same semantics (lemma 3.3),
- the synchronised product (for nodes) is compositional with respect to timed bisimulation (theorem 3.3),
- a timed node can be rewritten into a timed component that has the same semantics (theorem 3.4).

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.

Moreover the implementation of our translation called Timed AltaRica-Compiler is currently being added to the AltaRica toolbox.

Our future work is many-fold:

- complete the extension of AltaRica by adding features allowing the user to specify *hybrid systems* [18]; the main problem is to deal with time priorities in this case. The work we have presented in this paper is correct for *clock* variables but additional work is needed for systems where variables may have arbitrary integer slopes;
- study the problem of preserving liveness when using priorities (following the framework of [27]),
- use our Timed AltaRica-compiler on real industrial case studies,
- investigate in alternative ways of checking the correctness of Timed AltaRica specifications: this amounts to design some hierarchical model-checking algorithms taking advantage of the structure of Timed AltaRica specifications.

[12] Actually to prove this property with the restricted TCTL set of UPPAAL we have to change the initial set of states to check this property.

References

[1] G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[2] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[3] P. Le Guernic, M. Le Borgne, T. Gautier, and C. Le Maire. Programming real-time applications with Signal. *Proceedings of the IEEE*, 79(9):1321–1336, September 1991.

[4] F. Cassez and O. Roux. Compilation of the Electre reactive language into finite transition systems. *Theoretical Computer Science*, 146(1–2):109–143, juillet 1995.

[5] P. Pettersson and K. G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000.

[6] 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.

[7] 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.

[8] T. A. Henzinger, P. H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2):110–122, 1997.

[9] 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*.

[10] G. Point. *Altarica : Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement*. PhD thesis, University of Bordeaux I, Janvier 2000.

[11] A. Arnold, A. Griffault, G. Point, and A. Rauzy. The AltaRica formalism for describing concurrent systems. *Fundamenta Informaticae*, 40:109–124, 2000.

[12] 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.

[13] A. Arnold, D. Begay, and P. Crubille. *Construction and analysis of transition systems with MEC*. World Scientific, 1994.

[14] A. Arnold. *An experience with MEC in a real industrial project*. 1995.

[15] A. Vincent. *Conception et réalisation d'un vérificateur de modèles AltaRica*. PhD thesis, University of Bordeaux I, 2003.

[16] A. Griffault and A. Vincent. The mec 5 model checker. In *International Conference on Computer Aided Verification (CAV'04)*, Lecture Notes in Computer Science. Springer-Verlag, July 2004. to appear.

[17] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science B*, 126:183–235, 1994.

[18] T. A. Henzinger. The theory of hybrid automata. In *Proceedings, 11$^{\text{th}}$ Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.

[19] 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.

[20] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, W. Yi, and C. Weise. New generation of UPPAAL, 1998.

[21] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer-Verlag, 2001.

[22] 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.

[23] K. G. Larsen, P. Pettersson, and W. Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press, December 1995.

[24] 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.

[25] A. Arnold. *Finite transition systems*. Prentice-Hall, Masson, 1994.

[26] S. Bornot and J. Sifakis. On the composition of hybrid systems. In *HSCC*, pages 49–63, 1998.

[27] S. Bornot, G. Goessler, and J. Sifakis. On the construction of live timed systems. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 109–126, 2000.

[28] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. *Lecture Notes in Computer Science*, 1536:103–129, 1998.

[29] S. Bornot and J. Sifakis. An algebraic framework for urgency. *Information and Computation*, 163(1):172–202, 2000.

[30] J. Sifakis and S. Yovine. Compositional specification of timed systems (extended abstract). In *Symposium on Theoretical Aspects of Computer Science*, pages 347–359, 1996.

[31] 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.

[32] C. Pagetti. *Extension temporisée du langage AltaRica*. PhD thesis, Ecole Centrale de Nantes et Université de Nantes, April 2004.

A. Appendices

A.1. Proof of Theorem 3.1

Theorem A.1. Two TITS $\mathcal{A}_1$ and $\mathcal{A}_2$ are timed bisimilar if and only if there exists a TITS $\mathcal{B}$ and two timed interfaced bisimulation homomorphisms $h_1 : \mathcal{A}_1 \to \mathcal{B}$ and $h_2 : \mathcal{A}_2 \to \mathcal{B}$.

Proof:

If part. Assume there exist two homomorphisms $h_1 : \mathcal{A}_1 \longrightarrow \mathcal{B}$ and $h_2 : \mathcal{A}_2 \longrightarrow \mathcal{B}$ and denote $\mathcal{B} = \langle E_t, F_t, S_t, \pi, T\rangle$. Define the relation $R \subseteq S_{1_t} \times S_{2_t}$ by: $R(s, s') \iff h_1(s) = h_2(s')$ We show that R is a timed interfaced bisimulation by proving it satisfies the four points of Def. 3.2.

1. Let $q_1 \in S_{1_t}$ et $s = h_1(q_1) \in S_t$. Since h_2 is surjective there exists $q_2 \in S_{2_t}$ s.t. $h(q_2) = s = h(q_1)$.
2. Let $(q_1, q_2) \in R$. Then $h_1(q_1) = h_2(q_2)$. By item 2 of Def. 3.3 it follows that $\pi_1(q_1) = \pi(h_1(q_1)) = \pi(h_2(q_2)) = \pi_2(q_2)$.
3. Let $(q_1, g, e, q_1', g_1') \in T_1$. By item 3 of Def. 3.3 there exists g' s.t. $(h_1(q_1), g, e, h_1(q_1'), g') \in T$. Let $q_2 \in S_{2_t}$ s.t. $(q_1, q_2) \in R$. Then $h_2(q_2) = h_1(q_1)$ and by item 4 of Def. 3.3 we have i) there exists $q_2' \in S_{2_t}$ s.t. $h_2(q_2') = h_1(q_1')$ *i.e.* $(q_1', q_2') \in R$ and ii) there exists $g_2' \in \pi_2(q_2')$ s.t. $(q_2, g, e, q_2', g_2') \in T_2$. As the two TITS $\mathcal{A}_1$ and $\mathcal{A}_2$ play a symmetric role we obtain both items 3 and 4 of Def. 3.2.

Only if part The idea is as follows: there is a largest bisimulation $\equiv_1$ for $\mathcal{A}_1$ and $\equiv_2$ for $\mathcal{A}_2$ and those two largest bisimulations give two transition systems $\mathcal{A}_{1/\equiv_1}$ and $\mathcal{A}_{2/\equiv_2}$ that are isomorphic. Moreover each bisimulation $\equiv_i$ can be made a function and we can build two homomorphisms from these functions. The scheme is depicted in Fig. 12.

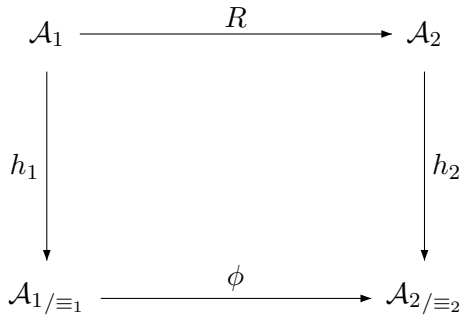


Figure 12. Building a Timed bisimulation Homomorphism

Let $V \subseteq X_1 \times X_2$ and $U \subseteq X_2 \times X_3$ be two binary relations. We denote $V.U \subseteq X_1 \times X_3$ the relation s.t. $(q, q') \in (V.U) \iff \exists q_2 \in X_2$ s.t. $(q, q_2) \in V$ and $(q_2, q') \in U$. For a binary relation $L \subseteq X \times X$ denote $L^{-1} = \{(q, q') | (q', q) \in L\}$ and $L^* = \cup_{n\in\mathbb{N}} L^n$ with $L^0 = Id$ and $L^{i+1} = L.L^i$. Note that $(L.L^{-1})^*$ is an equivalence relation.

Assume $\mathcal{A}_1$ and $\mathcal{A}_2$ are timed bisimilar and denote R a bisimulation relation on $S_{1_t} \times S_{2_t}$.

The relations $\equiv_1 \stackrel{def}{=} (R.R^{-1})^* \subseteq S_{1_t} \times S_{1_t}$ and $\equiv_2 \stackrel{def}{=} (R^{-1}.R)^* \subseteq S_{2_t} \times S_{2_t}$ are both equivalence relations. We define the quotient $\mathcal{A}_{1/\equiv_1}$ and $\mathcal{A}_{2/\equiv_2}$ and the functions $h_i : \mathcal{A}_i \longrightarrow \mathcal{A}_{i/\equiv_i}$ by: $h_i(s) = [s]$ ($[s]$ denotes the equivalence class for s). It is easy to see that h_i is a timed bisimulation homomorphism. Also define the mapping ϕ by:

$$\begin{array}{rccl} \phi: & \mathcal{A}_{1/\equiv_1} & \longrightarrow & \mathcal{A}_{2/\equiv_2} \\ & [s] & \longmapsto & [s'] \iff (s, s') \in R \end{array}$$

ϕ is a (timed bisimulation) isomorphism. Now let $h'_1 = \phi \circ h_1$. h'_1 is a timed bisimulation homomorphism (composition of an isomorphism and homomorphism). This completes the proof.

□

A.2. Proof of Theorem 3.2

Theorem A.2. (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 $h : \mathcal{A}_1|< \longrightarrow \mathcal{A}_2|<$ is also a timed bisimulation homomorphism.

Proof:
Let $h : \mathcal{A}_1 \longrightarrow \mathcal{A}_2$ be a timed bisimulation homomorphim. We show that h is also a timed bisimulation homomorphim from $\mathcal{A}_1|<$ onto $\mathcal{A}_2|<$.

For points 1 and 2 of Def. 3.3 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 $(q_1, g, e, q'_1, g') \in T_1|<$, then $(h(q_1), g, e, h(q'_1), g') \in T_2$. Assume that $(h(q_1), g, e, h(q'_1), g') \notin T_2|<$, then according to Def. 3.6 there are two possibilities:

1. either $e = t \in \mathbb{T}$ and $\exists e' >_0 time, t' < t, (h(q_1), g, t', q''_2, g'') \in T_2 \wedge (q''_2, g'', e', q'''_2, g''') \in T_2$. Since h is an homomorphism and $(h(q_1), g, t', q''_2, g'')$ and $(q''_2, g'', e'', q'''_2, g''')$ are in T_2, there exists $q''_1, q'''_1 \in S_{1_t}$ s.t. $h(q''_1) = q''_2, h(q'''_1) = q'''_2 \wedge (q_1, g, t', q''_1, g'')$ and $(q''_1, g'', e', q'''_1, g''')$ are in T_1. Hence (q_1, g, e, q'_1, g') cannot be in $T_1|<$ which contradicts the first assumption.

2. otherwise $e \in E_+$ and $\exists e'|e <_k e'$ and $\exists t \leq k|(h(q_1), g, t, q''_2, g'') \in T_2$ and $(q''_2, g'', e', q'''_2, g''') \in T_2$. Since h is an homomorphism and $(h(q_1), g, t, q''_2, g'')$ and $(q''_2, g'', e', q'''_2, g''')$ 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, g'') and $(q''_1, g'', e', q'''_1, g''')$ are in T_1. This contradicts again the fact that $(q_1, g, e, q'_1) \in T_1|<$. This ends the proof of point 3.

Now let $q_1 \in S_{1_t}, q'_2 \in S_{2_t}$ such that $(h(q_1), g, e, q'_2, g') \in T_2|<$. Then $\exists q'_1 \in S_{1_t}|h(q'_1) = q'_2$ and $(q_1, g, e, q'_1, g') \in T_1$. Again assume that for all q'_1 s.t. $h(q'_1) = q'_2$, we have $(q_1, g, e, q'_1, g') \notin T_1|<$:

1. if $e \in \mathbb{T}$, this means that $\exists t_1 < t, (q_1, g, t_1, q''_1, g'') \in T_1$ and $\exists e' >_0 time$ that is firable from (q''_1, g'). Then from item 4 of the Def. of homomorphism we get $(h(q_1), g, t_1, h(q''_1), g'') \in T_2 \wedge (h(q''_1), g'', e', q'''_2, g''') \in T_2$ and it contradicts $(h(q_1), g, e, q'_2, g') \in T_2|<$.

2. Otherwise $e \in E_+$ and there exists $e' \in E_+$ and $t \leq k$ s.t. $e <_k e'$, $(q_1, g, t, q''_1, g'') \in T_1$ and $(q''_1, g'', e', q'''_1, g''') \in T_1$. It follows that $(h(q_1), g, t, h(q''_1), g''), (h(q''_1), g'', e', h(q'''_1), g''') \in T_2$ which again contradicts the hypothesis $(h(q_1), g, e, q'_2) \in T_2|<$. This ends the proof of point 4 and of the theorem.

□

A.3. Proof of Theorem 3.3

Theorem A.3. Let $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\widetilde{V}, <_{\widetilde{V}})\rangle$ and $\mathcal{N}' = \langle V_F, C_F, E, <, \mathcal{N}'_0, \cdots, \mathcal{N}'_n, (\widetilde{V}, <_{\widetilde{V}})\rangle$ be two timed nodes such that $\forall i \in [0..n]$ there is a timed homomorphism h_i from $[\![\mathcal{N}_i]\!]$ to $[\![\mathcal{N}'_i]\!]$. Then there exists a timed homomorphism h from $[\![\mathcal{N}]\!]$ to $[\![\mathcal{N}']\!]$.

Proof:
Let $[\![\mathcal{N}_i]\!] = \langle E_t, F_{i_t}, S_{i_t}, \pi_i, T_i\rangle$ and $[\![\mathcal{N}'_i]\!] = \langle E_t, F_{i_t}, S'_{i_t}, \pi'_i, T'_i\rangle$ be the TITS that give the semantics of the timed nodes. First we assume $<$ is the empty relation.

Define h by:

$$\begin{array}{rrcl} h: & S_t & \longrightarrow & S'_t \\ & q = (q_0, \ldots, q_n) & \longmapsto & h(q) = (h_0(q_0), \ldots, h_n(q_n)) \end{array}$$

We prove that h is timed bisimulation homomorphism from $\mathcal{N}$ to $\mathcal{N}'$.

1. h is obviously surjective,
2. assume $m = |C_F|$ and $\overline{f} = (f, \mu)$. For $\pi(q)$ we get:

$$\begin{array}{rcl} \pi(q) & = & \{\overline{f} \in \mathcal{D}^{V_F} \times \mathbb{R}^m \mid \forall i \in [1,n],\, \exists \eta_i \in \pi_i(q_i) \mid (\overline{f}, \eta_1, \ldots, \eta_n) \in \pi_0(q_0)\} \\ & = & \{\overline{f} \in \mathcal{D}^{V_F} \times \mathbb{R}^m \mid \forall i \in [1,n],\, \exists \eta_i \in \pi'_i(h_i(q_i)) \mid (\overline{f}, \eta_1, \ldots, \eta_n) \in \pi'_0(h_0(q_0))\} \\ & = & \pi(h(q)) \end{array}$$

3. if $(q, g, e, q_1, g') \in T$ since $<$ is the empty relation and by definition of T (see Def. 3.10) it follows that $(h(q), g, e, h(q_1), g') \in T'$,
4. let $q = (q_0, q_1, \ldots, q_n) \in S_t$ and $q' = (q'_0, q'_1, \ldots, q'_n) \in S'_t$ s.t. $(h(q), g, e, q', g') \in T'$. Assume $e = (e_0, \ldots, e_n)$. Then by definition of T' (Def. 3.10) we have:

$$\begin{cases} \exists f_0 = (g, f_1, \ldots, f_n) \in \pi'_0(h_0(q_0)) \\ \exists f'_0 = (g', f'_1, \ldots, f'_n) \in \pi'_0(q') \\ \text{s.t. } \forall i \in [0,n] \quad (h_i(q_i), f_i, e_i, q'_i, f'_i) \in T'_i \end{cases}$$

As each h_i is an homomorphism we get: $\exists q''_i \in S_{i_t}$ s.t. $h_i(q''_i) = q'_i$ and $f'_i \in \pi_i(q''_i)$ and $(q_i, f_i, e_i, q''_i, f'_i) \in T_i$. This means:

$$\begin{cases} \exists f_0 = (g, f_1, \ldots, f_n) \in \pi_0(q_0) \\ \exists f'_0 = (g', f'_1, \ldots, f'_n) \in \pi'_0(q''_0) \\ \text{s.t. } \forall i \in [0,n] \quad (q_i, f_i, e_i, q''_i, f'_i) \in T_i \end{cases}$$

Take $q'' = (q''_0, \ldots, q''_n)$. We have $h(q'') = q'$ and $\exists f'_0 \in \pi(q'')$ s.t. $(q, g, e, q'', g') \in T$.

Now assume $<$ is not the empty relation. Let $\mathcal{N}_\emptyset = \langle V_F, C_F, E, \emptyset, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\widetilde{V}, <_{\widetilde{V}})\rangle$ and $\mathcal{N}'_\emptyset = \langle V'_F, C'_F, E, \emptyset, \mathcal{N}'_0, \cdots, \mathcal{N}'_n, (\widetilde{V}, <_{\widetilde{V}})\rangle$. With the previous proof in the case of an empty priority relation we get that there exists an homomorphism h from $\mathcal{N}_\emptyset$ to $\mathcal{N}'_\emptyset$. Applying theorem 3.2 we obtain that h is also an homomorphism from $\mathcal{N}$ to $\mathcal{N}'$. This completes the proof.

□

A.4. Proof of Theorem 3.4

Theorem A.4. Let $\mathcal{N}$ be a timed node. Then $\mathcal{N}$ can be rewritten into a timed component $\mathcal{C}_{\mathcal{N}}$ such that $[\![\mathcal{N}]\!]$ and $[\![\mathcal{C}_{\mathcal{N}}]\!]$ are timed bisimilar.

Proof:
We prove theorem 3.4 by induction. Let $\mathcal{N} = \langle V_F \cup C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$ and $[\![\mathcal{N}]\!] = \langle E_t, F_t, S_t, \pi, T\rangle$. Let $\mathcal{C}_{\mathcal{N}} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset\rangle$ as given by Definition 3.4, and $[\![\mathcal{C}_{\mathcal{N}}]\!] = \langle E_t, F_t, S'_t, \pi', T'\rangle$.

Base Step For the base step assume all the $\mathcal{N}_i$ are timed components: $\mathcal{N}_i = \langle V_{S_i} \cup C_{S_i}, V_{F_i} \cup C_{F_i}, E_i, A_i, M_i, \emptyset\rangle$ (note the priority relation is empty as we can use the syntactical encoding of timed priorities defined in section 3.7) and $[\![\mathcal{N}_i]\!] = \langle E_{i_t}, F_{i_t}, S_{i_t}, \pi_i, T_i\rangle$.

We prove that $\mathcal{N}$ and $\mathcal{C}_{\mathcal{N}}$ are timed bisimilar. Take equality as a candidate to be a bisimulation relation.

1. checking that equality is a total relation on $S_t \times S'_t$ amounts to checking that $\forall q \in S_t, \pi(q) = \pi'(q)$ and this is done in the second point,
2. let $q = (q_0, q_1, \cdots, q_n)$ and $\overline{f} = (f, \mu)$;

$$\begin{array}{rcl}
\pi(q) & = & \{\overline{f} \in \mathcal{D}^{V_F} \times \mathbb{R}^m \,|\, \forall i \in [1..n], \exists \eta_i \mid \eta_i \in \pi_i(q_i) \wedge (\overline{f}, \eta_1, \cdots, \eta_n) \in \pi_0(q_0)\} \\
& = & \{\overline{f} \in \mathcal{D}^{V_F} \times \mathbb{R}^m \,|\, \forall i \in [1..n], \exists \eta_i \mid (q_i, \eta_i) \in [\![A_i]\!] \wedge (q, \overline{f}, \eta_1, \cdots, \eta_n) \in [\![A_0]\!]\} \\
& = & \{\overline{f} \in \mathcal{D}^{V_F} \times \mathbb{R}^m \,|\, \forall i \in [1..n], \exists \eta_i \mid (q, \overline{f}, \eta_1, \cdots, \eta_n) \in [\![\wedge_{i=0..n} A_i]\!]\} \\
& = & \{\overline{f} \in \mathcal{D}^{V_F} \times \mathbb{R}^m \,|\, \forall i \in [1..n], \exists \eta_i \mid (q, \overline{f}) \in [\![\exists_{i=1..n}(V_{F_i} \cup C_{F_i}).\, \wedge_{i=0..n} A_i]\!]\} \\
& = & \{\overline{f} \in \mathcal{D}^{V_F} \times \mathbb{R}^m \,|\, \forall i \in [1..n], \exists \eta_i \mid (q, \overline{f}) \in [\![A]\!]\} \\
& = & \pi'(q)
\end{array}$$

3. let $\langle (q_0, q_1, \cdots, q_n), f, e, (q'_0, q'_1, \cdots, q'_n), f'\rangle \in T$ We denote $q = (q_0, q_1, \cdots, q_n)$ and $q' = (q'_0, q'_1, \cdots, q'_n)$. Note that $T_i = [\![M_i]\!]$ as for all $\mathcal{N}_i$ the priority relation is empty.
   - if $e = (e_0, e_1, \cdots, e_n)$ with $e_i \notin \mathbb{T}$ then by Def. 3.10:

$$\left.\begin{array}{l} \exists f_0 = (f, f_1, \cdots, f_n) \in \pi_0(q_0) \\ \exists f'_0 = (f', f'_1, \cdots, f'_n) \in \pi_0(q'_0) \end{array}\right\} \textit{ such that } \forall i \in [0..n], (q_i, f_i, e_i, q'_i, f'_i) \in [\![M_i]\!]$$

     This entails that $(q, f) \in [\![A]\!]$. Also by Def. 3.8 $\forall i \in [0..n]$ there exists a transition $((g_i, \gamma_i), e_i, (a_i, R_i)) \in M_i$ s.t. $(q_i, f_i) \in [\![A_i \wedge g_i \wedge \gamma_i]\!]$ and $q'_i = (a_i, R_i)(q_i, f_i)$ and $(q'_i, f'_i) \in [\![A_i]\!]$. This implies that $(q', f') \in [\![A]\!]$. It remains to find a transition in M s.t. the transition from (q, f) to (q', f') is fireable in $\mathcal{C}_{\mathcal{N}}$. Take $((g, \gamma), e, (a, R))$ in $\mathcal{C}_{\mathcal{N}}$ given by $g = (\exists_{i=1..n} V_{F_i}).g_0 \wedge \cdots \wedge g_n$, $\gamma = (\exists_{i=1..n} C_{F_i}).\gamma_0 \wedge \cdots \wedge \gamma_n$ and $a(x) = a_i(x)$ for $x \in V_S \cap V'_{S_i}$ and $R(c) = R_i(c)$ for $c \in C_S \cap C'_{S_i}$. As $\forall i \in [0..n]\, (q_i, f_i) \in [\![A_i \wedge g_i \wedge \gamma_i]\!]$ we have $(q, f) \in [\![\exists_{i=1..n} V_{F_i}).g_0 \wedge \cdots \wedge g_n \bigwedge \gamma = (\exists_{i=1..n} C_{F_i}).\gamma_0 \wedge \cdots \wedge \gamma_n]\!]$ and as $(q, f) \in [\![A]\!]$ we get $(q, f) \in [\![A \wedge g \wedge \gamma]\!]$. As we already mentioned $(q', f') \in [\![A]\!]$. Moreover $(q', f') =$

$(a, R)(q, f)$ and by Def. 3.8 this means that $\langle (q_0, q_1, \cdots, q_n), f, e, (q'_0, q'_1, \cdots, q'_n), f' \rangle \in T'$. The converse is straightforward and we finally have $\langle (q, f), e, (q', f') \rangle \in T \iff \langle (q, f), e, (q', f') \rangle \in T'$ for $e = (e_0, e_1, \cdots, e_n)$ with $e_i \notin \mathbb{T}$.

- if $e = (\delta, \delta, \cdots, \delta)$ with $\delta \in \mathbb{T}$ the have to ensure that all long the way from q to q' the time constraint (invariant) holds. This is straightforward by pointing out that if $(q, f), \delta, (q', f') \in T$ then $\forall \delta' \leq \delta$ we have $(q, f), \delta', (q'', f'') \in T$ for some (q'', f''). Then using Def. 3.8 we easily get the result that $(q_0, q_1, \cdots, q_n), f, \delta, (q'_0, q'_1, \cdots, q'_n), f' \rangle \in T'$. Again the converse holds and we end up with $T = T'$.

Induction Step Let $\mathcal{N} = \langle V_F \cup C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}}) \rangle$ and assume all the $\mathcal{N}_i$ can be re-written into timed components $\mathcal{C}_{\mathcal{N}_i}$ s.t. $\mathcal{N}_i$ and $\mathcal{C}_{\mathcal{N}_i}$ are timed bisimilar. Let $\mathcal{N}' = \langle V_F \cup C_F, E, <, \mathcal{C}_{\mathcal{N}0}, \cdots, \mathcal{C}_{\mathcal{N}n}, (\tilde{V}, <_{\tilde{V}}) \rangle$. Then using the base step proof we conclude there exists a timed component $\mathcal{C}_{\mathcal{N}'}$ that is bisimilar to $\mathcal{N}'$. By theorem 3.3 we also have that $\mathcal{N}'$ and $\mathcal{N}$ are timed bisimilar and hence $\mathcal{C}_{\mathcal{N}'}$ and $\mathcal{N}$ are timed bisimilar.

This completes the proof.

□

A.5. Proof of Theorem 3.5

Theorem A.5. Let $\mathcal{T} = (V_S, C_S, V_F, C_F, E, A, M, <)$ be a timed component where $>$ consists in one element: $e_u > \textit{time}$ (e_u is urgent). Assume there is one urgent transition $t_u = ((g_u, \gamma_u), e_u, (a_u, R_u))$ and γ_u is not sharp. Define the timed component $\mathcal{T}_u = (V_S, C_S, V_F, C_F, E, A \wedge \varphi_u, M, \emptyset)$ with

$$\varphi_u \stackrel{def}{=} g_u \implies \big((\gamma_u \wedge \neg(\gamma_u\uparrow)) \implies Y = 0\big) \tag{7}$$

Assume the flow variable Y is reset on each discrete transition and evolves at rate 1 on delay transitions. Then $[\![\mathcal{T}]\!]$ and $[\![\mathcal{T}_u]\!]$ are timed bisimilar.

Proof:
First note that as γ_u is not sharp $[\![\gamma_u \wedge \neg(\gamma_u\uparrow)]\!] \neq \emptyset$. Indeed for all $\epsilon > 0$ the set $W_\epsilon = \{\nu + \epsilon' \mid \nu \in [\![\gamma_u\uparrow]\!] \textit{ and } \epsilon' \leq \epsilon\}$ is included in $\neg(\gamma_u\uparrow)$. Hence if $[\![\gamma_u \wedge \neg(\gamma_u\uparrow)]\!] = \emptyset$ it must be the case that for all $\epsilon > 0$ the set $W_\epsilon \cap [\![\gamma_u]\!]$ is empty and contradicts the fact that γ_u is not sharp.

Second $\gamma_u \wedge \neg(\gamma_u\uparrow)$ is *past-opened i.e.* if $\nu \in [\![\gamma_u \wedge \neg(\gamma_u\uparrow)]\!]$ then there is $\epsilon > 0$ such that $\forall \epsilon' \leq \epsilon$ we have $\nu - \epsilon' \in [\![\gamma_u \wedge \neg(\gamma_u\uparrow)]\!]$. This follows from the constraint that the guard of an urgent transition has a first instant for which it becomes true.

Again we assume Y is not part of the configuration of the system and it is a global variable updated by an oracle. This enables us to use our notion of timed bisimilarity (Def. 3.2). Otherwise we would have to define timed bisimilarity for timed components with different sets of (clock) flows.

$[\![M]\!]\!|\!<$ is the transition relation of $[\![\mathcal{T}]\!]$ and we denote $[\![M_u]\!]$ the transition relation of $\mathcal{T}_u$.

First part: $[\![M]\!]\!|\!< \subseteq [\![M_u]\!]$. Remark that discrete transitions are unchanged and we only need to prove that each delay transition in $[\![M]\!]\!|\!<$ is a transition in $[\![M_u]\!]$.

Assume $(s, \nu), (f, \mu), \delta, (s, \nu'), (f, \mu') \in [\![M]\!]\!|\!<$ with $\delta > 0$. By Def. 3.8, item 4.(c) it implies that i) $(s, \nu), (f, \mu), \delta, (s, \nu'), (f, \mu') \in [\![M]\!]$ and by Def. 3.6 that ii) $\forall t' < t$ s.t. $((s, \nu), (f, \mu), t', (s, \nu +$

$t'), (f, \mu + t')) \in [\![M]\!]$, if $((s, \nu + t'), (f, \mu + t'), e, q', g') \in [\![M]\!]$ then $e \not> \textit{time}$. By i) we know that $\forall \delta' \leq \delta$ if $((s, \nu), (f, \mu), \delta', q', g') \in [\![M]\!]$ then $(q', g') \in [\![A]\!]$. Now if $(s, f) \in [\![g_u]\!]$ we need to prove that $(\nu', \mu') \in \varphi_u$. $\delta > 0$ implies $Y > 0$. Hence $(\gamma_u \wedge \neg(\gamma_u\uparrow))$ must be false at $(s, \nu'), (f, \mu')$. Assume it is true at $(s, \nu'), (f, \mu')$. Then as $(\gamma_u \wedge \neg(\gamma_u\uparrow))$ is past-opened there must be $\epsilon > 0$ s.t. $\forall \epsilon' \leq \epsilon$, $((s, \nu' - \epsilon'), (f, \mu' - \epsilon')) \in [\![(\gamma_u \wedge \neg(\gamma_u\uparrow))]\!]$. This contradicts ii). Thus $(\gamma_u \wedge \neg(\gamma_u\uparrow))$ is not satisfied for any $\delta' \leq \delta$ and $A \wedge \varphi_u$ holds all along the delay transition $(s, \nu), (f, \mu), \delta, (s, \nu'), (f, \mu')$ which implies this transition is in $[\![M_u]\!]$.

Second part: $[\![M_u]\!] \subseteq [\![M]\!]\!\!\upharpoonleft\!\!<$. Assume $(s, \nu), (f, \mu), \delta, (s, \nu'), (f, \mu') \in [\![M_u]\!]$ and $\delta > 0$. By Def. 3.8 this means that all long the path from $(s, \nu), (f, \mu)$ to $(s, \nu'), (f, \mu')$ assertion $A \wedge \varphi_u$ holds. In case $(s, f) \notin [\![g_u]\!]$ clearly the transition is in $[\![M]\!]\!\!\upharpoonleft\!\!<$. If $(s, f) \in [\![g_u]\!]$ then $(\gamma_u \wedge \neg(\gamma_u\uparrow)) \implies Y = 0$ must hold all along the way. Again as $Y > 0$ (because $\delta > 0$) it must be the case that $(\gamma_u \wedge \neg(\gamma_u\uparrow))$ is false for all $0 < t' \leq \delta$ *i.e.* $\neg\gamma_u \vee \gamma_u\uparrow$ holds. Either $\forall 0 < t' < \delta$, γ_u is false and in this case no urgent transition can be fired on the way from $(s, \nu), (f, \mu)$ to $(s, \nu'), (f, \mu')$ and $(s, \nu), (f, \mu), \delta, (s, \nu'), (f, \mu') \in [\![M]\!]\!\!\upharpoonleft\!\!<$. Or γ_u holds for some $0 < t'' < t$. Then $\gamma_u\uparrow$ must hold at t''. By the fact that $\gamma_u \wedge \gamma_u\uparrow$ is not single we know that there is a $t'' < t''' < t$ s.t. γ_u holds at t''' as well as $\neg(\gamma_u\uparrow)$). Hence Y must be equal to zero at t''' which cannot be the case. Hence there cannot be any $t'' < \delta$ s.t. t_u is fireable and again $(s, \nu), (f, \mu), \delta, (s, \nu'), (f, \mu') \in [\![M]\!]\!\!\upharpoonleft\!\!<$.
This completes the proof.

□

A.6. Proof of Theorem 4.1

Theorem A.1. Let $\mathcal{T} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, \emptyset\rangle$ be a timed component with $V_F = C_F = \emptyset$. Then $[\![\mathcal{T}]\!]$ and $[\![A(\mathcal{T})]\!]$ are timed bisimilar.

Note that we have shifted the clock flows into the state variables of the component and we allow only flows that are clock-definable.

Proof:
We denote $[\![\mathcal{T}]\!] = \langle E_t, F_t, S_t, \pi, T\rangle$ with $F_t = \{tt\}$ (F_t cannot be empty because of the definition of TITS) and π is constant and equal to $\pi(q) = tt$. $A(\mathcal{T}) = (L, L_0, E, X, I, T)$ and the semantics of $A(\mathcal{T})$ is a TTS (see section 3.1) $(Q, E, \rightarrow)$. Thus it is also TITS $\langle E'_t, F'_t, S'_t, \pi', T'\rangle$ of dimension $(|X|, 0)$, with $E'_t = E \cup \mathbb{R}_{\geq 0}$, $F_t = \{tt\}$, $S'_t = \{(l, \nu) \mid \nu \in [\![inv(l)]\!]\}$, $\pi(q) = tt$ (always non empty) and T' is given by $\rightarrow$. We omit f is configuration like (s, f) as it always amounts to (s, tt).Now we prove that $[\![\mathcal{T}]\!]$ and $[\![A(\mathcal{T})]\!]$ are timed bisimilar: as a candidate for a timed bisimulation relation we take equality of the states.

1. Let $q = (s, \nu) \in S_t$. We apply Def. 3.8:

$$
\begin{array}{rcl}
(s, \nu) \in S_t & \iff & (s, \nu) \in [\![A_{V_T} \wedge A_{C_T}]\!] \\
& \iff & s \in [\![A_{V_T}]\!] \wedge (s, \nu) \in [\![A_{C_T}]\!] \\
& \iff & s \in l^L_G,\ G \uplus L = [1..p] (l^L_G \textit{ form a partition of } A_{V_T}) \textit{ and } \nu \in [\![\cap_{k \in G} I_k]\!] \\
& \iff & s \in l^L_G \wedge \nu \in [\![Inv(l^L_G)]\!] \\
& \iff & (s, \nu) \in S'_t
\end{array}
$$

2. π and π' are constant and equal to *tt*, so they agree for each state,

3. let $(s,\nu) \in S_t$. Then $(s,\nu) \in S'_t$ by item 1 above. Assume $((s,\nu),e,(s_1,\nu_1)) \in T$ with $e \in E_+$. We have to prove that $((s,\nu),e,(s_1,\nu_1)) \in T'$. As $((s,\nu),e,(s_1,\nu_1)) \in T$ there is a transition $t = (g,\gamma),e,(a,R)) \in M$ such that i) $(s,\nu) \in [\![A \wedge g \wedge \gamma]\!]$, ii) $s_1 = a(s)$ and iii) $\nu_1 = R(\nu)$. This implies that in $A(\mathcal{T})$ there is a transition of the form $(l_G^L, g \wedge \mathit{Pre}_t(l_{G_1}^{L_1}) \wedge \gamma, e, (a,R), l_{G_1}^{L_1})$. By iii) we get that $(s,\nu) \in [\![\mathit{Pre}_t(l_{G_1}^{L_1})]\!]$. By i) (s,ν) satisfies $g \wedge \gamma \wedge l_G^L$. So in the semantics of $A(\mathcal{T})$ a transition of the form $((s,\nu),e,(s',\nu'))$ can be taken. As we use (a,R) to update the values of the state variables and (a,R) are deterministic we obtain $(s',\nu') = (s_1,\nu_1)$. Now assume $((s,\nu),\delta,(s_1,\nu_1)) \in T$ with $\delta \in \mathbb{R}_{\geq 0}$. By definition $(s,\nu) \in [\![A]\!]$ and $(s_1,\nu_1) \in [\![A]\!]$ and as we have time determinism for the flow variables $\forall 0 \leq \delta' \leq \delta$, $(s,\nu+\delta') \in I(s)$ with $I(s) = \cap_{k|s\in P_k} I_k$. $s \in l_G^L$ fo some $G \uplus L = [1..p]$ and the invariant for l_G^L is $Inv(l_G^L) = \cap_{k\in G} I_k$. $Inv(l_G^L)$ and $I(s)$ coincides and thus $((s,\nu),\delta,(s_1,\nu_1)) \in T'$.

4. the converse of item 3 above is straightforward and proved exactly as item 3.

This completes the proof. □