A calculus for timed automata
(Extended abstract)*

Pedro R. D'Argenio and Ed Brinksma

Dept. of Computer Science. University of Twente.
P.O.Box 217. 7500 AE Enschede. The Netherlands.
{*dargenio,brinksma*}*@cs.utwente.nl*

Abstract. A language for representing timed automata is introduced. Its semantics is defined in terms of timed automata. This language is complete in the sense that any timed automaton can be represented by a term in the language. We also define a direct operational semantics for the language in terms of (timed) transition systems. This is proven to be equivalent (or, more precisely, timed bisimilar) to the interpretation in terms of timed automata.
In addition, a set of axioms is given that is shown to be sound for timed bisimulation. Finally, we introduce several features including the parallel composition and derived time operations like wait, time-out and urgency. We conclude with an example and show that we can eliminate non-reachable states using algebraic techniques.

1 Introduction

A real-time system is a system whose behaviour is constrained by requirements on the time in which events can occur. Sometimes, systems are implemented as timed systems in the sense that they fulfil certain timing conditions to give them an acceptable performance. Other systems depend on timing conditions in a more essential way, viz. because their functional correctness depends upon certain critical timing conditions being fulfilled. Therefore, it becomes interesting to study the formal verification of such systems.

In the last years, several formal techniques have been developed to specify and verify real-time systems. For instance, many well-known process algebras have been extended with features to manipulate time [11, 26, 22, 23, 5, 17, 6, 9, 18]. But the apparently most successful approaches are timed and hybrid automata [3, 24, 15, 2]. The formal relation between these two models has been studied in some cases [24, 25, 14, 12]. Languages that fully represent timed automata have also been studied [20, 27].

In this paper, we introduce a process algebra to describe timed automata. Since the syntax of timed automata becomes unwieldy to specify realistic real-time systems, the process algebra introduced here proposes a higher-level language that is interpreted in terms of timed automata. More specifically, we choose

* Supported by the NWO/SION project 612-33-006.

a slight variation of the so called timed safety automata [15]. Basically, the language extends Milner's CCS [21] restricted to prefixing, inaction and summation, with some features to manipulate clocks, namely, clock resetting, invariants and guards. We prove that any timed automaton can be described by a term in the language together with guarded recursion.

Also, we introduce a direct operational semantics for the language. Thus, a (timed) transition system is associated to each process. We prove that this way of giving semantics is equivalent (*timed bisimilar*) to the interpretation of the associated timed automaton.

In order to facilitate the construction of complex systems we include parallel composition, and several common operations on time, such as time-out, waiting and urgency.

The first goal of our paper is to introduce a powerful language to represent timed automata. Our second goal is to introduce an equational theory for the language that allows us to manipulate timed automata in order to eliminate redundant information. This is an interesting point, since, to our knowledge, timed automata have not yet been studied from an algebraic point of view. The axiomatisation is sound for timed bisimulation and allows to find a normal form. Moreover, the additional operators like parallel composition can be eliminated, thus obtaining equivalent expressions defined just in terms of the basic language.

As an example we study the railroad crossing controller of [3]. In this example, we illustrate that we can eliminate redundant states, clocks, and conditions. In particular, non-reachable states are eliminated.

The rest of this paper is structured as follows. Section 2 reviews the models of timed transition systems and timed automata. In Section 3, we introduce the language and we study its relation with timed automata. The operational semantics is introduced in Section 4 and the relation with the timed automata model is stated. Section 5 introduces the axiomatisation for the basic language, and the extension with new operators is studied in Section 6. The example is presented in Section 7. Extensionality with respect to CCS, related work, and conclusions are discussed in Section 8.

This article is an extended abstract of [10] which contains all proofs omitted here.

Acknowledgements. This work profited from discussions with Jan F. Groote, Rom Langerak, Jan Springintveld, Jan Tretmans, Frits Vaandrager and Sergio Yovine. In particular, Sergio Yovine pointed out the related work [27]. Reference [4] was pointed out by one of the referees.

2 Models for Timed Systems

Time, clocks and constraints. We adopt the set $\mathbb{R}^{\geq 0}$ of non-negative reals as *time domain*. A *clock* is a variable x ranging over a time domain $\mathbb{R}^{\geq 0}$. Let $\mathcal{C}$ denote a set of clocks. The set $\Phi(\mathcal{C})$ of *clock constraints* over $\mathcal{C}$ is defined inductively by:

$$\phi ::= \quad d \leq d' \;\mid\; x \leq d \;\mid\; d \leq x \;\mid\; x - y \leq d \;\mid\; d \leq x - y \;\mid\; (\phi \wedge \phi) \;\mid\; (\neg \phi)$$

where $d, d' \in \mathbb{R}^{\geq 0}$ and $x, y \in \mathcal{C}$ with $x \neq y$. The abbreviations **tt**, **ff**, $x = d$, $x > d$, $x - y < d$, $\phi \vee \phi'$, $x \in [d, d')$, etc. are defined as usual. Let $var(\phi)$ denote the set of clocks occurring in ϕ. A clock constraint is closed if no clocks occur in it. We denote the set of closed clock constraints by Φ^c. We could also adopt a richer set of constraints (see Section 8).

A *(clock) valuation* is a function $v : \mathcal{C} \to \mathbb{R}^{\geq 0}$. Let $\mathcal{V}$ denote the set of valuations. v is lifted to clocks constraints by the obvious induction over the structure of ϕ. Let $C \in \mathcal{C}$. We define $v[C \hookleftarrow 0]$ as $v[C \hookleftarrow 0](x) \stackrel{\text{def}}{=}$ **if** $x \in C$ **then** 0 **else** $v(x)$. Let $d \in \mathbb{R}^{\geq 0}$. Define $v + d$ as $(v + d)(x) \stackrel{\text{def}}{=} v(x) + d$. Notice that for any valuation v and for any clock constraint ϕ, $v(\phi)$ is a closed clock constraint.

For the subset of closed clock constraints, we define the satisfaction predicate $\models \subseteq \Phi^c$ as usual:

$$\frac{}{\models d \leq d + d'} \qquad \frac{\models \phi \quad \models \phi'}{\models (\phi \wedge \phi')} \qquad \frac{\not\models \phi}{\models (\neg \phi)}$$

where $d, d' \in \mathbb{R}^{\geq 0}$. We generalise $\models$ to all clock constraints ($\models \subseteq \Phi(\mathcal{C})$). Let $\phi \in \Phi(\mathcal{C})$ then $\models \phi \stackrel{\text{def}}{\iff} \forall v \in \mathcal{V} : \models v(\phi)$.

We define the set $\overline{\Phi}(\mathcal{C}) \subseteq \Phi(\mathcal{C})$ of *past-closed constraints* as $\phi \in \overline{\Phi}(\mathcal{C}) \stackrel{\text{def}}{\iff} \models (v + d)(\phi) \implies \models v(\phi)$, for all $v \in \mathcal{V}$ and $d \in \mathbb{R}^{\geq 0}$. Notice that this kind of constraints are such that if they hold at time d, they hold at all $d' < d$.

Timed Transition Systems. A timed transition system is a labelled transition system that includes information about the time. We adopt the model of actions with *time stamps*.

Definition 1. Let $\mathbf{A}$ be a set of *actions*. A *timed transition system* (TTS) is a structure $L = (S, \mathbf{A} \times \mathbb{R}^{\geq 0}, s_0, \longrightarrow, \mathcal{U})$ where

- S is a set of *states*, with the *initial state* $s_0 \in S$;
- A is a set of *labels*;
- $\longrightarrow \subseteq S \times (\mathbf{A} \times \mathbb{R}^{\geq 0}) \times S$ is the *transition relation*; and
- $\mathcal{U} \subseteq \mathbb{R}^{\geq 0} \times S$ is the *until predicate*.

We use the following notation: $a(d)$ iff $(a, d) \in \mathbf{A} \times \mathbb{R}^{\geq 0}$, $s \xrightarrow{a(d)} s'$ iff $\langle s, a(d), s' \rangle \in \longrightarrow$, $\mathcal{U}_d(s)$ iff $\langle d, s \rangle \in \mathcal{U}$, $s \xrightarrow{a(d)}$ iff $\exists s' \in S.\ s \xrightarrow{a(d)} s'$.

In addition, L should satisfies the following axioms:

Until $\quad \forall d, d' \in \mathbb{R}^{\geq 0}.\ \mathcal{U}_d(s) \wedge d' < d \implies \mathcal{U}_{d'}(s)$;

Delay $\quad \forall d \in \mathbb{R}^{\geq 0}.\ s \xrightarrow{a(d)} \implies \mathcal{U}_d(s)$. □

The intended meaning of a transition $s \xrightarrow{a(d)} s'$ is that a system which is in state s can change to be in state s' by performing an action a at time d. Intuitively, $\mathcal{U}_d(s)$ together with axiom **Until**, means that a system can idle in a state s at least d units of times. Axiom **Delay** state that every time that an action may occur in a state s at time d, the system must be idling at that time.

Predicate $\mathcal{U}$ was introduced in [17]. Here, we formalised its behaviour in a relative time setting by adding the axioms **Until** and **Delay**.

Definition 2. Let $L_i = (S_i, \mathbf{A} \times \mathbb{R}^{\geq 0}, s_0^i, \longrightarrow_i, \mathcal{U}^i)$, $i \in \{1,2\}$, be two TTS. A *timed bisimulation* is a relation $R \subseteq S_1 \times S_2$ with $s_0^1 R s_0^2$ satisfying, for all $a(d) \in \mathbf{A} \times \mathbb{R}^{\geq 0}$, the following transfer properties:

1. if $s_1 R s_2$ and $s_1 \xrightarrow{a(d)}_1 s_1'$, then $\exists s_2' \in S_2 :\ s_2 \xrightarrow{a(d)}_2 s_2'$ and $s_1' R s_2'$;
2. if $s_1 R s_2$ and $s_2 \xrightarrow{a(d)}_2 s_2'$, then $\exists s_1' \in S_1 :\ s_1 \xrightarrow{a(d)}_1 s_1'$ and $s_1' R s_2'$; and
3. if $s_1 R s_2$, then $\mathcal{U}_d^1(s_1) \iff \mathcal{U}_d^2(s_2)$.

If such a relation exists, we say that L_1 and L_2 are *timed bisimilar* (notation $L_1 \underline{\leftrightarrow} L_2$). □

Timed Automata. In this paragraph we define a variation of timed automata [3]. We use invariants as in [15, 24, 25] but, instead of considering clock resettings on the edges, we consider them in the states. The reason for this is that we want to avoid assumptions about the initial setting of clocks, which makes the compositionality of the language more complicated. Compare [27] (see Section 8).

Definition 3. A *timed (safety) automaton* is a structure $(S, \mathbf{A}, \mathcal{C}, s_0, \longrightarrow, \partial, \kappa)$ where:

- S is a set of *states*, with the *initial state* $s_0 \in S$;
- $\mathbf{A}$ is a set of *actions*;
- $\mathcal{C}$ is a set of *clocks*;
- $\longrightarrow \subseteq S \times \mathbf{A} \times \Phi(\mathcal{C}) \times S$ is the set of *edges*;
- $\partial : S \to \overline{\Phi}(\mathcal{C})$ is the *invariant assignment* function;
- $\kappa : S \to \wp_{\mathrm{fin}}(\mathcal{C})$ is the *clocks resetting* function.

The set of all timed automata is denoted by $\mathcal{T}$. □

In this case, $\langle s, a, \phi, s' \rangle \in \longrightarrow$ (notation $s \xrightarrow{a,\phi} s'$) intuitively means that when the system is in state s it could change to be in state s' by performing an action a provided that the clock constraint ϕ holds. The clock setting function states which clocks should be reset as soon as a state is reached. The invariant assignment function states that the system can idle in a state s as long as $\partial(s)$ holds.

Notice that our timed automata can be translated into timed automata with resettings on the edges by just labelling the edge with the set of clocks to be reset in the target state, that is, an edge $s \xrightarrow{a,\phi} s'$ will be translated into $s \xrightarrow{a,\phi,\kappa(s')} s'$. Conversely, a timed automaton with resettings on the edges could be transformed by "pushing" the clock resetting into the target state, i.e., given an edge $s \xrightarrow{a,\phi,C} s'$ we define $s \xrightarrow{a,\phi} s'$ and $\kappa(s') \stackrel{\mathrm{def}}{=} C$. In case that many edges with different clock resettings go to the same state, this state is "split" into different states, one for each set of clocks.

Formally speaking, a timed automaton can be interpreted as a TTS as follows.

Definition 4. Let $T = (S, \mathbf{A}, \mathcal{C}, s_0, \longrightarrow, \partial, \kappa) \in \mathcal{T}$ be a timed automaton. Let $v_0 \in \mathcal{V}$ be any valuation. The *interpretation* of T with initial valuation v_0 is given by the TTS $[\![T]\!]_{v_0} \stackrel{\mathrm{def}}{=} (S \times \mathcal{V}, \mathbf{A} \times \mathbb{R}^{\geq 0}, (s_0, v_0), \longrightarrow, \mathcal{U})$ where $\longrightarrow$ and $\mathcal{U}$ are defined as the least sets satisfying the following rules:

$$\frac{s \xrightarrow{a,\phi} s' \quad \models (v[\kappa(s)\hookleftarrow 0] + d)(\phi \wedge \partial(s)) \qquad \models (v[\kappa(s)\hookleftarrow 0] + d)(\partial(s))}{(s,v) \xrightarrow{a(d)} (s', (v[\kappa(s)\hookleftarrow 0] + d)) \qquad \mathcal{U}_d(s,v)}$$

□

Since $\partial(s) \in \overline{\Phi}(\mathcal{C})$ for all $s \in S$, it follows that $(\!|T|\!)_{v_0}$ satisfies axiom **Until**. Moreover, notice that if $(s,v) \xrightarrow{a(d)}$ then $\models (v[\kappa(s)\hookleftarrow 0] + d)(\partial(s))$ and so $\mathcal{U}_d(s,v)$ which implies that axiom **Delay** holds. Hence, $(\!|T|\!)_{v_0}$ is indeed a TTS for any initial valuation v_0.

Isomorphism is a fine enough equivalence. Thus, proving the existence of an isomorphism is enough to prove that two timed automata are equivalent in coarser equivalences, for instance, timed bisimulation.

Definition 5. Let $T = (S, \mathbf{A}, \mathcal{C}, s_0, \longrightarrow, \partial, \kappa)$ and $T' = (S', \mathbf{A}, \mathcal{C}, s'_0, \longrightarrow', \partial', \kappa')$ be two timed automata. An *isomorphism* from T to T' is a bijective function $\Gamma : S \to S'$ such that

1. $\Gamma(s_0) = s'_0$,
2. $s \xrightarrow{a,\phi} s' \iff \Gamma(s) \xrightarrow{a,\phi}{}' \Gamma(s')$,
3. $\partial(s) = \partial'(\Gamma(s))$, and
4. $\kappa(s) = \kappa'(\Gamma(s))$.

We say that T and T' are *isomorphic*, notation $T \cong T'$, if there is an isomorphism between T and T'. □

3 A Simple Language for Timed Automata

In this section we introduce a simple language that contains the necessary operators to represent timed automata. We give the semantics of this language in terms of timed automata. Moreover, we show that any timed automaton could be represented by a term of this language if we add guarded recursion over expressions.

Definition 6. Let $\mathbf{A}$ be a set of actions and let $\mathcal{C}$ be a set of clocks. The language $\mathcal{L}$ is defined according to the following grammar:

$$p ::= \quad \mathbf{stop} \mid a;p \mid \phi \mapsto p \mid p + p \mid \{\!|C|\!\}\, p \mid \psi \mathrel{\triangleright\!\!>} p$$

where $a \in \mathbf{A}$, $\phi \in \Phi(\mathcal{C})$, $\psi \in \overline{\Phi}(\mathcal{C})$ and $C \in \wp_{\mathrm{fin}}(\mathcal{C})$. We refer to the elements of $\mathcal{L}$ as processes. □

Process **stop** represents inaction; it is the process that cannot perform any action. The intended meaning of $a;p$ (named *(action-)prefixing*) is that action a can be performed at any time and then it behaves like p. $\phi \mapsto p$, the *guarding operation*, executes any first action that p can do whenever ϕ holds. $\{\!|C|\!\}\, p$, the *clock resetting operation*, is a process that behaves like p, but resetting the clocks in C. We will write $\{\!|x_1, \ldots, x_n|\!\}\, p$ instead of $\{\!|\{x_1, \ldots, x_n\}|\!\}\, p$. $\psi \mathrel{\triangleright\!\!>} p$, the

invariant operation, can idle while ψ holds or go on with the process p. $p+q$ is the *choice*; it executes either p or q. The choice between p and q can be made only by actions, not by the passage of time.

Definition 7. Let $p \in \mathcal{L}$. The set $fv(p)$ of *free variables* of p and the set $bv(p)$ of *bound variables* of p are defined as the least set satisfying

$$\begin{array}{lll} fv(\mathbf{stop}) & = \emptyset & bv(\mathbf{stop}) = \emptyset \\ fv(a;p) & = fv(p) & bv(a;p) = bv(p) \\ fv(\phi{\mapsto}p) & = var(\phi) \cup fv(p) & bv(\phi{\mapsto}p) = bv(p) \\ fv(p+q) & = fv(p) \cup fv(q) & bv(p+q) = bv(p) \cup bv(q) \\ fv(\{\!|C|\!\}\, p) & = fv(p) \backslash C & bv(\{\!|C|\!\}\, p) = C \cup bv(p) \\ fv(\psi \triangleright\!\!> p) & = var(\psi) \cup fv(p) & bv(\psi \triangleright\!\!> p) = bv(p) \end{array}$$

□

Notice that the term $\{\!|C|\!\}\, p$ binds clocks in C that appear in any constraints in p. Let K_p be the union of all clock resettings in p which do not occur within the scope of a prefixing, i.e., a subterm $a;q$. We say that a term *does not have conflict of variables* if there is no subterm in it that has conflict of variables and, if it has the form $p+q$ (respectively $\psi \triangleright\!\!> p$) then $(fv(p) \cap K_q) \cup (fv(q) \cap K_p) = \emptyset$ (respectively $var(\psi) \cap K_p = \emptyset$). In this work we will generally assume processes which do not have conflict of variables. This assumption is harmless since we can always rename properly bound variables (i.e. to apply $\alpha-$conversion) in order to avoid this problem. In [10] we study $\alpha-$conversion and conflict of variables extensively.

We can associate a timed automaton to a process according to the following definition.

Definition 8. Let $p \in \mathcal{L}$ be without conflict of variables. The *timed automaton associated to* p is defined by $[\![p]\!]^{\mathcal{T}} = (\mathcal{L}, \mathbf{A}, \mathcal{C}, p, \longrightarrow, \partial, \kappa)$ where $\longrightarrow$, ∂ and κ are defined as the least sets satisfying the rules of Table 1. □

The notion of associated timed automaton is well-defined for processes without conflict of variables. In order to check it, we should see that for all $q \in \mathcal{L}$, ∂ and κ are indeed functions and moreover, that $\partial(q) \in \overline{\Phi}(\mathcal{C})$. But it can be straightforwardly proven by induction on the depth of the proof tree taking into account that if $\psi, \psi' \in \overline{\Phi}(\mathcal{C})$ then $\psi \wedge \psi', \psi \vee \psi' \in \overline{\Phi}(\mathcal{C})$.

Rules in Table 1 capture the behaviour above described in terms of timed automata. In particular, it deserves to notice that a process $p+q$ can idle as long as one of them can. Thus $\partial(p+q) \iff \partial(p) \vee \partial(q)$. Moreover, $p+q$ can execute any action of p or q as long as it could be executed in its original process. Thus, since an action cannot be executed after the idling time is finished, we require that for the execution of an action, the corresponding invariant must also hold.

The condition that processes should not have conflict of variables is necessary. If it were not considered we would have undesirable bindings. For instance, consider the term $p \equiv (x \leq 2) \triangleright\!\!> (\{\!|x|\!\}\ (x=1){\mapsto}a;\mathbf{stop})$. Clearly, x is free in the invariant $(x \leq 2)$, however, using rules in Table 1, we derive $\partial(p) = (x \leq 2)$ and

Table 1. Timed automata for $\mathcal{L}$

$\kappa(\mathbf{stop}) = \emptyset$	$\kappa(a;p) = \emptyset$	$\dfrac{\kappa(p) = C}{\kappa(\phi \mapsto p) = C}$
$\dfrac{\kappa(p) = C \quad \kappa(q) = C'}{\kappa(p+q) = (C \cup C')}$	$\dfrac{\kappa(p) = C'}{\kappa(\{\!\|C\|\!\}\ p) = (C \cup C')}$	$\dfrac{\kappa(p) = C}{\kappa(\psi \rhd\!\!> p) = C}$
$\partial(\mathbf{stop}) = \mathbf{tt}$	$\partial(a;p) = \mathbf{tt}$	$\dfrac{\partial(p) = \psi}{\partial(\phi \mapsto p) = \psi}$
$\dfrac{\partial(p) = \psi \quad \partial(q) = \psi'}{\partial(p+q) = (\psi \vee \psi')}$	$\dfrac{\partial(p) = \psi}{\partial(\{\!\|C\|\!\}\ p) = \psi}$	$\dfrac{\partial(p) = \psi'}{\partial(\psi \rhd\!\!> p) = (\psi \wedge \psi')}$
$a; p \xrightarrow{a,\mathbf{tt}} p$	$\dfrac{p \xrightarrow{a,\phi'} p'}{\phi \mapsto p \xrightarrow{a,\phi \wedge \phi'} p'}$	$\dfrac{p \xrightarrow{a,\phi} p' \quad \partial(p) = \psi}{p+q \xrightarrow{a,\phi \wedge \psi} p' \quad q+p \xrightarrow{a,\phi \wedge \psi} p'}$
$\dfrac{p \xrightarrow{a,\phi} p'}{\{\!\|C\|\!\}\ p \xrightarrow{a,\phi} p'}$	$\dfrac{p \xrightarrow{a,\phi} p'}{\psi \rhd\!\!> p \xrightarrow{a,\phi} p'}$	

$\kappa(p) = \{x\}$. Thus, according to Definition 4 the x in the invariant is captured by the clock resetting. Similar reasoning shows that, in $q \equiv ((y \leq 1) \rhd\!\!> a; \mathbf{stop}) + (\{\!|y|\!\}\ \mathbf{stop})$, the free occurrence of y in the left operand is captured by the clock resetting in the right operand since $\partial(q) = (y \leq 1)$ and $\kappa(q) = \{y\}$.

We extend the expressiveness of our language by allowing recursive specifications.

Definition 9. Let $\mathbf{V}$ be a set of *process variables*. We extend the previous language with process variables. So, let $\mathcal{L}^{\mathbf{v}}$ the language defined by the following grammar:

$$p ::= \quad \mathbf{stop} \ \mid \ a;p \ \mid \ \phi \mapsto p \ \mid \ p+p \ \mid \ \{\!\|C\|\!\}\ p \ \mid \ \psi \rhd\!\!> p \ \mid \ X$$

where $a \in \mathbf{A}$, $\phi \in \Phi(\mathcal{C})$, $\psi \in \overline{\Phi}(\mathcal{C})$, $C \subseteq \mathcal{C}$ and $X \in \mathbf{V}$. A *recursive specification* is a set of *recursive equations* having the form $X = p(\mathbf{V})$ for each $X \in \mathbf{V}$, where $p(\mathbf{V}) \in \mathcal{L}^{\mathbf{v}}$. Every recursive specification has a distinguished process variable called *root*. We extend the notion of free and bound variables by adding the equations $fv(X) = fv(q)$ and $bv(X) = bv(q)$ provided $X = q \in E$. □

We recall that $fv(p)$ and $bv(p)$ are defined as the least set satisfying the equations in Definitions 6 and 9. Thus, for instance, if $X = \{\!\|x\|\!\}\ (y \leq 3 \wedge x < 2) \rhd\!\!> X$ then $fv(X) = \{y\}$ and $bv(X) = \{x\}$.

Now, we extend the notion of the associated timed automaton to recursive specifications and we state the correctness of the definition.

Definition 10. Let E be a recursive specification such that none of its equations has conflict of variables. The *timed automaton associated to* $p \in \mathcal{L}^{\mathbf{v}}$ is defined

by $[\![p]\!]^{\mathcal{T}} = (\mathcal{L}, \mathbf{A}, \mathcal{C}, p, \longrightarrow, \partial, \kappa)$ where $\longrightarrow$, ∂ and κ are defined as the least set satisfying rules in Table 1 and rules in Table 2. □

Table 2. Timed automata for recursion ($X = p \in E$)

$\dfrac{\kappa(p[p/X]) = C}{\kappa(X) = C}$	$\dfrac{\partial(p[p/X]) = \psi}{\partial(X) = \psi}$	$\dfrac{p[p/X] \xrightarrow{a,\phi} p'}{X \xrightarrow{a,\phi} p'}$

Definition 11. An *occurrence of X is guarded* in a term $p \in \mathcal{L}^{\mathbf{v}}$ if p has a subterm $a;q$ such that this occurrence of X is in q. A term p is *guarded* if all the occurrences of its variables are guarded. A recursive specification is *guarded* if the right hand side of every recursive equation in it is a guarded process. □

Notice that ∂ and κ are not always well-defined in case of (unguarded!) recursion. For instance, take $X = (x < 1) \rrbracket X$, then ∂ and κ are the completely undefined functions because of nonterminating derivation. Nevertheless, we can state the following theorem.

Theorem 12. *Let $p \in \mathcal{L}^{\mathbf{v}}$ be a process without conflict of variables, which has process variables defined in a guarded recursive specification E without conflict of variables. The associated timed automaton $[\![p]\!]^{\mathcal{T}}$ is indeed a timed automaton.*

Proof. It can be proved by structural induction that ∂ and κ are defined for any guarded term. In addition, we can see that for all $q \in \mathcal{L}^{\mathbf{v}}$ relations ∂ and κ are functions and moreover, that $\partial(q) \in \overline{\Phi}(\mathcal{C})$. This can be proven by induction on the derivation of ∂ and κ. □

The language presented here, together with a guarded recursive specification, has the property of expressing any timed automaton in the sense of Theorem 13 below. First, we borrow some definitions from transition system theory into timed automaton theory. A timed automaton is *image-finite* if the set of outgoing edges of every state labelled with the same action is finite, i.e., for any a and any s, the set $\{s \xrightarrow{a,\phi} s' |\ s' \in S\}$ is finite. It is *finitely sorted* if, for each state s, the set of all actions labelling the outgoing edges, i.e., $\{a |\ \exists s' \in S.\ s \xrightarrow{a,\phi} s'\}$ is finite. A state s is *(symbolically) reachable* if there is a sequence of edges from the initial state s_0 to s, i.e., there are $a_1, \ldots, a_n$, $\phi_1, \ldots, \phi_n$ and $s_1, \ldots, s_n$ ($n \geq 0$) such that $s_0 \xrightarrow{a_1,\phi_1} s_1 \cdots \xrightarrow{a_n,\phi_n} s_n = s$. The *reachable part* of a timed automaton T is the same timed automaton restricted to the set of states that are reachable. Notice that we are considering a static view but not the usual notion of reachability in timed automata theory (compare to [1]).

Theorem 13 Representability of timed automata. *For every image-finite and finitely sorted $T \in \mathcal{T}$ there is a guarded recursive specification E with root X_{s_0} such that the reachable part of T and the reachable part of $[\![X_{s_0}]\!]^{\mathcal{T}}$ are isomorphic.*

Proof. The proof consists of associating a process variable to each state s of T and defining each one of them as the term that resets the clocks of $\kappa(s)$ and has an invariant $\partial(s)$ over the summation of the outgoing edges represented by prefixings with its respective guard. Thus, the isomorphism is given by the function that maps every state in its corresponding variable.

Let $T = (S, \mathbf{A}, \mathcal{C}, s_0, \longrightarrow', \partial', \kappa')$. For each state $s \in S$ define a different variable X_s. Let V_S be the set of such variables. Define the set of recursive specifications E with root X_{s_0} and recursive equations

$$X_s = \{\!|\kappa'(s)|\!\}\ \partial'(s) \circledcirc \left(\sum \{\phi \mapsto a; X_{s'} \mid s \xrightarrow{a,\phi} s'\}\right)$$

where $\sum\{p_i \mid i \in \{1 \ldots n\}\} \stackrel{\text{def}}{=} p_1 + p_2 + \cdots + p_n$. In particular, if s has no outgoing transition then $X_s = \{\!|\kappa'(s)|\!\}\ \partial'(s) \circledcirc \mathbf{stop}$.

According to Definition 10, $[\![X_{s_0}]\!]^{\mathcal{T}} = (\mathcal{L}^{\mathbf{v}}, \mathbf{A}, \mathcal{C}, X_{s_0}, \longrightarrow, \partial, \kappa)$. Define

$$[\![X_{s_0}]\!]^{\mathcal{T}} \restriction V_S \stackrel{\text{def}}{=} (V_S, \mathbf{A}, \mathcal{C}, X_{s_0}, \longrightarrow \restriction V_S, \partial \restriction V_S, \kappa \restriction V_S)$$

where:
$$\longrightarrow \restriction V_S \stackrel{\text{def}}{=} \longrightarrow \cap (V_S \times \mathbf{A} \times \Phi(\mathcal{C}) \times \mathcal{L}^{\mathbf{v}})$$
$$\partial \restriction V_S \stackrel{\text{def}}{=} \partial \cap (V_S \times \overline{\Phi}(\mathcal{C}))$$
$$\kappa \restriction V_S \stackrel{\text{def}}{=} \kappa \cap (V_S \times \wp(\mathcal{C}))$$

Clearly, $\Gamma : S \to V_S$ defined as $\Gamma(s) \stackrel{\text{def}}{=} X_s$ for all $s \in S$, is an isomorphism, which straightforwardly implies the theorem. □

In the previous proof, the restriction of $[\![X_{s_0}]\!]^{\mathcal{T}}$ to the set V_S is merely formal, and it is due to the fact that the associated timed automaton is defined considering the whole set of terms $\mathcal{L}^{\mathbf{v}}$ instead of the reachables ones.

In order to represent data it could be needed to consider more general timed automata which are not necessarily image finite or finitely sorted. This kind of automata could be represented in the language by defining an infinite summation operator in the expected way. Thus, Theorem 13 could be extended to timed automata with denumerable branching and denumerable sorts.

4 An Operational Semantics

In this section we give a semantics for $\mathcal{L}^{\mathbf{v}}$ in terms of TTS. We state that it coincides (modulo timed bisimulation) with the semantics of the associated timed automaton.

Definition 14. Let E be a recursive specification with process variables $\mathbf{V}$. The TTS of a term $p \in \mathcal{L}^{\mathbf{v}}$ with initial valuation $v_0 \in \mathcal{V}$ is defined by $(\!|p|\!)^*_{v_0} \stackrel{\text{def}}{=} (\mathcal{L}^{\mathbf{v}} \times \mathcal{V}, \mathbf{A} \times \mathbb{R}^{\geq 0}, (p, v_0), \longrightarrow, \mathcal{U})$ where $\longrightarrow$ and $\mathcal{U}$ are the least set satisfying rules in Table 3. □

Table 3. Operational semantics for $\mathcal{L}^{\mathbf{v}}$ ($a \in \mathbf{A}$, $d \in \mathbb{R}^{\geq 0}$, $v \in \mathcal{V}$, $X = p \in E$)

$$\mathcal{U}_d(\mathbf{stop}, v) \qquad \mathcal{U}_d(a; p, v) \qquad \frac{\mathcal{U}_d(p, v)}{\mathcal{U}_d(\phi \mapsto p, v)} \qquad \frac{\mathcal{U}_d(p, v[C \leftarrow 0])}{\mathcal{U}_d(\{\!|C|\!\}\ p, v)}$$

$$\frac{\models (v + d)(\psi) \quad \mathcal{U}_d(p, v)}{\mathcal{U}_d(\psi \triangleright\!\!> p, v)} \qquad \frac{\mathcal{U}_d(p, v)}{\mathcal{U}_d(p + q, v) \quad \mathcal{U}_d(q + p, v)} \qquad \frac{\mathcal{U}_d(p[p/X], v)}{\mathcal{U}_d(X, v)}$$

$$(a; p, v) \xrightarrow{a(d)} (p, v + d) \qquad \frac{\models (v + d)(\phi) \quad (p, v) \xrightarrow{a(d)} (p', v')}{(\phi \mapsto p, v) \xrightarrow{a(d)} (p', v')}$$

$$\frac{(p, v[C \leftarrow 0]) \xrightarrow{a(d)} (p', v')}{(\{\!|C|\!\}\ p, v) \xrightarrow{a(d)} (p', v')} \qquad \frac{\models (v + d)(\psi) \quad (p, v) \xrightarrow{a(d)} (p', v')}{(\psi \triangleright\!\!> p, v) \xrightarrow{a(d)} (p', v')}$$

$$\frac{(p, v) \xrightarrow{a(d)} (p', v')}{(p + q, v) \xrightarrow{a(d)} (p', v') \quad (q + p, v) \xrightarrow{a(d)} (p', v')} \qquad \frac{(p[p/X], v) \xrightarrow{a(d)} (p', v')}{(X, v) \xrightarrow{a(d)} (p', v')}$$

$(\!|p|\!)^*_{v_0}$ is well defined, that is, $(\!|p|\!)^*_{v_0}$ is indeed a TTS satisfying, thus, axioms **Until** and **Delay**, which can be proven by straightforward induction on the length of the proof tree.

The rules in Table 3 express the intended behaviour of each term in terms of TTS. In this case the execution of a transition or the idling time is made concrete. Thus, for instance, process $\phi \mapsto p$ can actually perform any action a that p can perform at time d in the valuation v whenever the condition ϕ holds in the valuation v after d units of time has passed. Or, on the other hand, process $\psi \triangleright\!\!> p$ can idle d units of times in a valuation v if p also can idle d units of time, and moreover, condition ψ holds in the valuation v after d units of time.

Now, we extend the notion of timed bisimilarity to the terms in the language.

Definition 15. Two terms $p, q \in \mathcal{L}$ are *timed bisimilar*, notation $p \underline{\leftrightarrow} q$, if and only if for all $v_0 \in \mathcal{V}$, $(\!|p|\!)^*_{v_0} \underline{\leftrightarrow} (\!|q|\!)^*_{v_0}$. □

We state that all the operators of the language preserve timed bisimulation, that is $\underline{\leftrightarrow}$ is a congruence in $\mathcal{L}$, which can be proven as usual.

So far we stated two ways of interpreting a term in $\mathcal{L}$. Function $(\!|\ |\!)^*$ associates a TTS to each term and closed valuation. On the other hand, a TTS could be associated to every $p \in \mathcal{L}$ in two steps, namely, by associating a timed automaton to p (see Definition 8) and then interpreting such a timed automaton in terms of a TTS according to Definition 4. Theorem 16 states that both way of interpreting a process are equivalent according to timed bisimulation.

Theorem 16. *Let E be a guarded recursive specification with process variables $\mathbf{V}$. For every $p \in \mathcal{L}^{\mathbf{v}}$ without conflict of variables and for every closed valuation v_0, $(\!|p|\!)^*_{v_0} \underline{\leftrightarrow} (\!|[\![p]\!]^{\mathcal{T}}|\!)_{v_0}$.*

Proof (Sketch). Assume $(\!|p|\!)^*_{v_0} = (\mathcal{L}^{\mathbf{v}} \times \mathcal{V}, \mathbf{A} \times \mathbb{R}^{\geq 0}, (p, v_0), \longrightarrow, \mathcal{U})$ and $(\!|[\![p]\!]^{\mathcal{T}}|\!)_{v_0}$ $= (\mathcal{L}^{\mathbf{v}} \times \mathcal{V}, \mathbf{A} \times \mathbb{R}^{\geq 0}, (p, v_0), \longrightarrow', \mathcal{U}')$. We state that $R \stackrel{\text{def}}{=} \{((q, v), (q, \overline{v}))\mid q \in \mathcal{L} \wedge v\restriction fv(q) = \overline{v}\restriction fv(q)\}$, with $v\restriction C \stackrel{\text{def}}{=} v \cap (C \times \mathbb{R}^{\geq 0})$, is a timed bisimulation. As usual, it is proven by verifying that the transfer properties hold. □

A summary of the studied relations is given in Figure 1, where arrows may be read as "can be interpreted in".

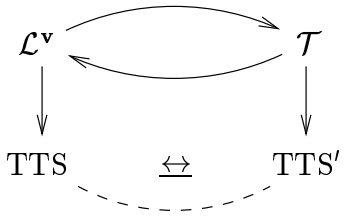

Fig. 1. Summary

5 Axiomatisation

In this section we give a set of axioms that holds in bisimulation models. It follows immediately that they also hold in any coarser model as for instance the several timed bisimulations with abstraction [26, 17], timed trace preorder and timed simulations [19, 20]. We consider terms modulo α-conversion without loss of generality.

Axioms in Table 4 could be explained as follows. The choice is commutative **A1** and associative **A2**. Axioms **A3** and **A3′** state a kind of idempotency of $+$ and **A4** states that **stop** is the neutral element for $+$ in the context of unbounded idling. **Stp** states that a prefixed process which does not satisfies its guard condition cannot proceed with its execution. Axioms **G0**–**G5** state the way in which guards can be simplifyed. Notice that they cannot be eliminated except in the case of **tt**. In particular, axioms **G3**, **G4** and **G5** say how to move invariants, clock resettings and summations out of the scope of a guard. Similarly, axioms **I1**–**I5** state how to simplify the invariant operation. **I3** says how to take clocks resettings out of the scope of an invariant, while **I4** and **I5** move the invariant out of the scope of a summation. **R1** and **R2** eliminate redundant clocks. In particular, **R2** implies that it is always possible to reduce the amount of clocks to be reset to at most one for each clock resetting operation. **R3** gathers all the clocks resettings in only one operation and **R4** moves clocks out of the scope of a summation. The term $[x \leftarrow y](p)$, which appears in axiom **R2**, is the renaming of the free occurrences of x by y in p. It is defined recursively on the structure of p in the obvious way. Finally, **D1** and **D2** state that the difference between clocks is invariant and thus it could be "transported" along the execution. In particular,

Table 4. Axioms for $\mathcal{L}$ ($a, b \in \mathbf{A}$, $C \subseteq \mathcal{C}$, $x, y \in \mathcal{C}$, $\phi, \phi' \in \Phi(\mathcal{C})$, $\psi, \psi' \in \overline{\Phi}(\mathcal{C})$, $d \in \mathbb{R}^{\geq 0}$)

Stp	$\mathbf{ff} \mapsto a; p = \mathbf{stop}$	
A1	$p + q = q + p$	
A2	$(p + q) + r = p + (q + r)$	
A3	$\phi \mapsto p + \phi' \mapsto p = (\phi \vee \phi') \mapsto p$	
A3'	$\psi \boxRightarrow p + \psi' \boxRightarrow p = (\psi \vee \psi') \boxRightarrow p$	
A4	$a; p + \mathbf{stop} = a; p$	
G0	$\phi \mapsto \mathbf{stop} = \mathbf{stop}$	
G1	$\mathbf{tt} \mapsto p = p$	
G2	$\phi \mapsto (\phi' \mapsto p) = (\phi \wedge \phi') \mapsto p$	
G3	$\phi \mapsto (\psi \boxRightarrow p) = \psi \boxRightarrow (\phi \mapsto p)$	
G4	$\phi \mapsto (\{\!\{C\}\!\}\ p) = \{\!\{C\}\!\}\ (\phi \mapsto p)$	if $\mathit{var}(\phi) \cap C = \emptyset$
G5	$\phi \mapsto (p + q) = \phi \mapsto p + \phi \mapsto q$	
I1	$\mathbf{tt} \boxRightarrow p = p$	
I2	$\psi \boxRightarrow (\psi' \boxRightarrow p) = (\psi \wedge \psi') \boxRightarrow p$	
I3	$\psi \boxRightarrow (\{\!\{C\}\!\}\ p) = \{\!\{C\}\!\}\ (\psi \boxRightarrow p)$	if $\mathit{var}(\psi) \cap C = \emptyset$
I4	$\psi \boxRightarrow p + \psi \boxRightarrow q = \psi \boxRightarrow (p + q)$	
I5	$\psi \boxRightarrow (\phi \mapsto a; p) + \psi' \boxRightarrow (\phi' \mapsto b; q) = (\psi \vee \psi') \boxRightarrow ((\psi \wedge \phi) \mapsto a; p + (\psi' \wedge \phi') \mapsto b; q)$	
R1	$\{\!\{C\}\!\}\ p = p$	if $C \cap \mathit{fv}(p) = \emptyset$
R2	$\{\!\{C \cup \{y, x\}\}\!\}\ p = \{\!\{C \cup \{y\}\}\!\}\ [x \leftarrow\!\shortmid y](p)$	
R3	$\{\!\{C\}\!\}\ \{\!\{C'\}\!\}\ p = \{\!\{C \cup C'\}\!\}\ p$	
R4	$\{\!\{C\}\!\}\ p + \{\!\{C\}\!\}\ q = \{\!\{C\}\!\}\ (p + q)$	
D1	$\phi \mapsto a; (\{\!\{y\}\!\}\ p) = \phi \mapsto a; (\{\!\{y\}\!\}\ (x - y \Box d) \boxRightarrow p)$	if $\models (\phi \Rightarrow (x \Box d))$ and $x \neq y$
D2	$\phi \mapsto a; p = \phi \mapsto a; ((x - y \Box d) \boxRightarrow p)$	if $\models (\phi \Rightarrow (x - y \Box d))$ where $\Box \in \{\leq, <, \geq, >, =\}$

D1 explains how this difference is stated. Notice that axioms do not necessarily preserve free variables. For instance, **G1** allows us to prove $(x \geq 0) \mapsto p = p$.

Some interesting properties that can be derived from the axioms (and induction when necessary) are idempotency of summation ($p + p = p$), invariants act also as guards ($\psi \boxRightarrow p = \psi \boxRightarrow (\psi \mapsto p)$) aand $\mathbf{ff} \boxRightarrow \mathbf{stop}$ is the neutral element for $+$ (i.e. $\mathbf{ff} \boxRightarrow \mathbf{stop} + p = p$).

Notice that $\mathbf{ff} \boxRightarrow a; p = \mathbf{ff} \boxRightarrow (\mathbf{ff} \mapsto a; p) = \mathbf{ff} \boxRightarrow \mathbf{stop}$ but $\mathbf{ff} \boxRightarrow \mathbf{stop} \not\leftrightarrows \mathbf{stop}$. This is due to the fact that timed bisimulation can model the halting of the progress of time. It could be understood as a broken machine that is not longer allowed to remain in the same state and, simultaneously, has no way to leave such a state, i.e., no action can be performed in order to leave such a state. This phenomenon is known as *time deadlock*. The difference with the ordinary deadlock phenomenon is that a system is in deadlock if it reaches a state that cannot perform any action, but such a state need not have any restrictions on idling, which is the case for time deadlock.

Axioms in Table 4 are sound for timed bisimulation as it is stated as follows.

Theorem 17 Soundness. *For all $p, q \in \mathcal{L}^{\mathbf{v}}$, if $p = q$ is deduced by means of*

equational reasoning using axioms in Table 4, then $p \underline{\leftrightarrow} q$.

An interesting property that is derived from these axioms is that every term can be expressed in a normal form.

Definition 18. Define the set $B \subseteq \mathcal{L}$ of *basic terms* inductively as follows:

- $\mathbf{stop} \in B'$
- $p \in B$, $\phi \in \Phi(\mathcal{C})$ and $a \in \mathbf{A} \implies \phi \mapsto a; p \in B'$
- $p, q \in B' \implies p + q \in B'$
- $p \in B'$, $\psi \in \overline{\Phi}(\mathcal{C})$ and $x \in \mathcal{C} \implies \{\!|x|\!\}\ \psi \mathrel{\mathbb{P}\!\!>} p \in B$

B' is the set of all terms whose clock resettings and invariants are all within the scope of a prefix construction. Notice that a basic term has the general format (modulo **A1**, **A2**, **A3** and **A4**)

$$p = \{\!|x|\!\}\ \psi \mathrel{\mathbb{P}\!\!>} \left(\textstyle\sum_{i \in I} \phi_i \mapsto a_i; p_i\right)$$

where each p_i is already a basic term. We adopt the convention that

$$\textstyle\sum_{i \in \emptyset} \phi_i \mapsto a_i; p_i = \mathbf{stop}$$ □

The following theorem can be proven by structural induction using the axioms.

Theorem 19. *For every term $p \in \mathcal{L}$ there is a term $q \in B$ such that $p = q$ can be proven by means of axioms in Table 4 and α-conversion.*

6 Other Operators

In this section, we introduce parallel composition and several well-known time operations such as wait, time-out and urgency.

Time Operations. In this paragraph we give some axiomatic definitions for common operations on time. The operation $\mathsf{wait}_d(p)$ waits d units of time before starting to execute p. Conversely, $\mathsf{before}_d(p)$ forces to execute p before d units of time have passed. They can be defined as follows, provided $x \notin fv(p)$:

$$\mathsf{wait}_d(p) \stackrel{\text{def}}{=} \{\!|x|\!\}\ (x \geq d) \mapsto p \qquad \mathsf{before}_d(p) \stackrel{\text{def}}{=} \{\!|x|\!\}\ (x \leq d) \mathrel{\mathbb{P}\!\!>} p$$

Analogously, we can define strict versions of the operators. In particular, we will consider $\mathsf{before}^{<}_d(p) \stackrel{\text{def}}{=} \{\!|x|\!\}\ (x < d) \mathrel{\mathbb{P}\!\!>} p$.

Urgency is defined by the operation $\mathsf{urgent}_d(p)$ that obliges to execute p just after waiting d units of time:

$$\mathsf{urgent}_d(p) \stackrel{\text{def}}{=} \mathsf{before}_d(\mathsf{wait}_d(p))$$

More generally we can define the operation $\mathsf{between}[d, d'](p)$ which forces the execution of p after waiting d units of time but before d' units of time have passed:

$$\mathsf{between}[d, d'](p) \stackrel{\text{def}}{=} \mathsf{before}_{d'}(\mathsf{wait}_d(p))$$

We can easily generalise this operation to open intervals in the obvious way.

Maybe, the most well known operation is the time-out. $p\,\mathsf{timeout}_d q$ forces to execute q just after waiting d units of time if process p does not started execution yet:

$$p\,\mathsf{timeout}_d q \stackrel{\text{def}}{=} \mathsf{before}_d^{<}(p) + \mathsf{urgent}_d(q)$$

Parallel Operator. We define a LOTOS-like parallel operator [8]. Basically, the process $p||_A q$ executes process p and q in parallel and forces synchronisation on actions in set $A \in \mathbf{A}$. $\between_A$ and $|_A$ are the left and communication merge respectively, which are needed to give a finite axiomatisation of the parallel operator. In order to define associated timed automata we will require the auxiliary operator $\overline{\mathsf{ck}}$ which is intended to avoid clocks resettings.

Table 5. Timed automata for the parallel operator

$$\frac{\kappa(p) = C \quad \kappa(q) = C'}{\begin{array}{c}\kappa(p||_A q) = (C \cup C') \\ \kappa(p \between_A q) = (C \cup C') \\ \kappa(p|_A q) = (C \cup C')\end{array}} \qquad \frac{\partial(p) = \psi \quad \partial(q) = \psi'}{\begin{array}{c}\partial(p||_A q) = (\psi \wedge \psi') \\ \partial(p \between_A q) = (\psi \wedge \psi') \\ \partial(p|_A q) = (\psi \wedge \psi')\end{array}}$$

$$\frac{p \xrightarrow{a,\phi} p'}{\begin{array}{l}p||_A q \xrightarrow{a,\phi} p'||_A \overline{\mathsf{ck}}(q) \\ q||_A p \xrightarrow{a,\phi} \overline{\mathsf{ck}}(q)||_A p' \\ p \between_A q \xrightarrow{a,\phi} p'||_A \overline{\mathsf{ck}}(q)\end{array}}\ a \notin A \qquad \frac{p \xrightarrow{a,\phi} p' \quad q \xrightarrow{a,\phi'} q'}{\begin{array}{l}p||_A q \xrightarrow{a,\phi\wedge\phi'} p'||_A q' \\ p|_A q \xrightarrow{a,\phi\wedge\phi'} p'||_A q'\end{array}}\ a \in A$$

$$\kappa(\overline{\mathsf{ck}}(p)) = \emptyset \qquad \frac{\partial(p) = \psi}{\partial(\overline{\mathsf{ck}}(p)) = \psi} \qquad \frac{p \xrightarrow{a,\phi} p'}{\overline{\mathsf{ck}}(p) \xrightarrow{a,\phi} p'}$$

Free and bound variables are defined by $fv(p \Box q) = fv(p) \cup fv(q)$ and $bv(p \Box q) = bv(p) \cup bv(q)$ for $\Box \in \{||_A, \between_A, |_A\}$, and $fv(\overline{\mathsf{ck}}(p)) = \kappa(p) \cup fv(p)$ and $bv(\overline{\mathsf{ck}}(p)) = bv(p)$. We say that $p||_A q$, $p \between_A q$ and $p|_A q$ do not have conflict of variables if neither p or q do and $(bv(p) \cap var(q)) \cup (bv(q) \cap var(p)) = \emptyset$.

We give the rules for the timed automaton in Table 5. Operators $\between_A$ and $|_A$ are the left-merge and the communicating versions of the parallel operator, respectively. Operation $\overline{\mathsf{ck}}$ is needed since if we admitted an edge like $p||_A q \xrightarrow{a,\phi} p'||_A q$ instead of $p||_A q \xrightarrow{a,\phi} p'||_A \overline{\mathsf{ck}}(q)$, the clocks of q, which were reset as soon as $p||_A q$ was reached, would be reset again when $p'||_A q$ is reached after performing action a. This last situation would be incorrect since the time for process q would then not have progressed.

Axioms for parallel composition are given in Table 6. Operator $\overline{\mathsf{ck}}$ is just required in order to define associated timed automata. Moreover, it does not preserve $\leftrightarrow$ and α-conversion. Thus, we are not interested in giving any axiomatisation of it. However, the information introduced for $\overline{\mathsf{ck}}$ is somehow encoded

in the axiomatisation by the operator B'. Notice that $\mathsf{B}'(p)$ holds when $p \in B'$ according to Definition 18, i.e. whenever no clock resetting or invariant appears out of the scope of a prefixing.

Table 6. Axioms for parallel composition

PC	$p\|\|_A q = p \between_A q + q \between_A p + p\|_A q$	
LM1	$\mathbf{stop} \between_A (\psi \boxRight q) = \psi \boxRight \mathbf{stop}$	if $\mathsf{B}'(q)$
LM2	$a; p \between_A (\psi \boxRight q) = \psi \boxRight \mathbf{stop}$	if $a \in A \wedge \mathsf{B}'(q)$
LM3	$a; p \between_A (\psi \boxRight q) = \psi \boxRight a; (p\|\|_A (\psi \boxRight q))$	if $a \notin A \wedge \mathsf{B}'(q)$
LM4	$(\phi \mapsto p) \between_A q = \phi \mapsto (p \between_A q)$	
LM5	$(p + q) \between_A r = p \between_A r + q \between_A r$	
LM6	$(\{\!\{C\}\!\}\, p) \between_A q = \{\!\{C\}\!\}\, (p \between_A q)$	if $C \cap fv(q) = \emptyset$
LM7	$(\psi \boxRight p) \between_A q = \psi \boxRight (p \between_A q)$	
LM8	$p \between_A \{\!\{C\}\!\}\, q = \{\!\{C\}\!\}\, (p \between_A q)$	if $C \cap fv(p) = \emptyset$
CM0	$p\|_A q = q\|_A p$	
CM1	$\mathbf{stop}\|_A \mathbf{stop} = \mathbf{stop}$	
CM2	$\mathbf{stop}\|_A a; p = \mathbf{stop}$	
CM3	$a; p\|_A a; q = a; (p\|\|_A q)$	if $a \in A$
CM4	$a; p\|_A b; q = \mathbf{stop}$	if $a \neq b \vee a \notin A$
CM5	$\phi \mapsto p\|_A q = \phi \mapsto (p\|_A q)$	
CM6	$(p + q)\|_A r = p\|_A r + q\|_A r$	
CM7	$(\{\!\{C\}\!\}\, p)\|_A q = \{\!\{C\}\!\}\, (p\|_A q)$	if $C \cap fv(q) = \emptyset$
CM8	$(\psi \boxRight p)\|_A q = \psi \boxRight (p\|_A q)$	

$\mathsf{B}'(\mathbf{stop})$ $\quad$ $\mathsf{B}'(a; p)$ $\quad$ $\dfrac{\mathsf{B}'(p)}{\mathsf{B}'(\phi \mapsto p)}$ $\quad$ $\dfrac{\mathsf{B}'(p)\ \ \mathsf{B}'(q)}{\mathsf{B}'(p + q)}$

It can be proven that for every term $p\|\|_A q$, $p \between_A q$ and $p\|_A q$ there is an α-convertible term without conflict of variables. Thus, for terms with conflict of variables we just assume their interpretation is the timed automata of some α-conversion without conflict of variables. Moreover, timed bisimulation is a congruence for $\|\|_A$, $\between_A$ and $\|_A$. We state that axioms are sound for timed bisimulation and they allow the elimination of these new operators.

Theorem 20 Soundness. *For all p and q obtained by extending $\mathcal{L}^{\mathbf{v}}$ with $\|\|_A$, $\between_A$ and $\|_A$, if $p = q$ is deduced by means of equational reasoning using axioms in Table 4 and axioms in Table 6, then $p \leftrightarrows q$.*

Theorem 21 Elimination. *For every term p in the language $\mathcal{L}$ extended with $\|\|_A$, $\between_A$ and $\|_A$, there is a q in $\mathcal{L}$ such that $p = q$ can be derived from axioms in Table 4 and Table 6.*

7 Example

We take the example of the automatic controller of a gate at a railroad crossing using the definition from [3], except that we have adapted it to include invariants. The components of the system can be described as follows. A *TRAIN* communicates to the controller that it *appr*oaches at least 2 minutes before it enters the crossing (*in*). After leaving the crossing (*out*), the *TRAIN* informs the *CONTROLLER* that it *exit*ed within 5 minutes after sending the signal *appr*. The *GATE* system receives the information when to *lower* the gate. This should be put *down* before 1 minute has passed. Then, the system waits for an order to *raise* the gate. After that, it is lifted (*up*) within 1 to 2 minutes. The *CONTROLLER* waits for a train to *appr*oach. After exactly 1 minute, it orders to *lower* the gate. Then, it waits until the train *exit*s the crossing and at most 1 minute afterwards it orders to *raise* the gate.

The components of the system can be described as follows.

$$
\begin{aligned}
TRAIN &= appr; \{\!|x|\!\}\ \big(\ (x<5) \trianglerighteq (x>2) \mapsto in; \\
&\qquad\qquad\qquad\quad (x<5) \trianglerighteq out; \\
&\qquad\qquad\qquad\quad (x<5) \trianglerighteq exit; TRAIN\ \big) \\
GATE &= lower;\ \mathsf{before}_1^{<}(down; raise;\ \mathsf{between}(1,2)(up; GATE)) \\
CONTROLLER &= appr;\ \mathsf{urgent}_1(lower; exit;\ \mathsf{before}_1^{<}(raise; CONTROLLER)) \\
SYSTEM &= CONTROLLER ||_{\{appr,exit,lower,raise\}} (TRAIN ||_{\emptyset} GATE)
\end{aligned}
$$

By using axioms in Table 6 parallel operations can be eliminated. Assuming only one clock for each component, the expression obtained at this point will contain 3 clocks and 19 states. However, many of those states are not reachable since the system will never meet conditions which allow that. These states can be eliminated by using axioms in Table 4, using **D1** and **D2** in particular. Moreover, the number of clocks can be reduced to 2. In this way, the *SYSTEM* can be proven equivalent to the following recursive specification which has 2 clocks and 10 states.

$$
\begin{aligned}
&S_0 = appr; S_1 && S_5 = (x<5) \trianglerighteq exit; S_6 \\
&S_1 = \{\!|x|\!\}\ S_1' && S_6 = \{\!|y|\!\}\ (y<1) \trianglerighteq raise; S_7 \\
&S_1' = (x \le 1) \trianglerighteq (x=1) \mapsto lower; S_2 && S_7 = \{\!|y|\!\}\ (y<2) \trianglerighteq (appr; S_8 \\
&S_2 = \{\!|y|\!\}\ (y<1) \trianglerighteq down; S_3 && \qquad\qquad + (y>1) \mapsto up; S_0) \\
&S_3 = (x<5) \trianglerighteq (x>2) \mapsto in; S_4 && S_8 = \{\!|x|\!\}\ (y<2 \wedge x \le 1) \trianglerighteq (y>1) \mapsto up; S_1' \\
&S_4 = (x<5) \trianglerighteq out; S_5
\end{aligned}
$$

Clock x keeps track of the evolution of the time with respect to the *TRAIN* and some activities in the *CONTROLLER* (particularly the action *lower*), while y keeps track of the time of the proper activities of the *GATE* (namely *down* and *up*) and the activity of *rais*ing the gate. Notice, however, that the action *up* in S_8 is also constrained by clock x (viz. the condition $x \le 1$). This would seem to imply that the *CONTROLLER* also controls the time of lifting the gate (action

up). Clearly, this is not a desirable situation. In [10] we consider an alternative example that avoids this problem.

The timed automaton associated with S_0 is depicted in Figure 2. States are represented by circles and their numbers are written beside. κ and ∂ are respectively written in the upper and lower part of the circle. Edges are represented by the arrows. Empty sets and true conditions are omitted, and singleton sets are represented by their elements.

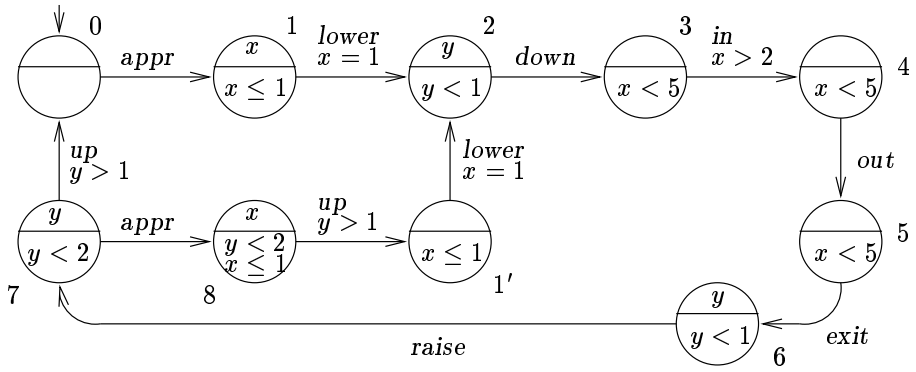

Fig. 2. The reduced timed automaton of the railroad crossing system

8 Further Remarks

Milner's Synchronisation Trees and our Language. Basically, our calculus is an extension of Milner's synchronisation trees [21] (i.e., CCS with only prefixing, inaction and summation) with operations to manipulate clocks (clock resettings, invariants and guards). Moreover, we can state that our calculus is an operational conservative extension up to (timed) bisimulation, that is, for every pair of terms obtained by using only prefixing, stop and summation (the *untimed terms*), they are (strong) bisimilar if and only if they are timed bisimilar.

Furthermore, the equational theory given for $\mathcal{L}$ (see Table 4) is an equational conservative extension of the equational theory for synchronisation trees (i.e. commutativity, associativity, idempotency and **stop** as neutral element of $+$). Thus, for each equality $p = q$ of untimed terms that can be proven in Milner's theory, it can also be proven in our theory and vice versa.

Related Works. Nicollin, Sifakis & Yovine [24, 25] give an interpretation of ATP [23] in terms of timed automata with invariants, considering a dense time domain. Such a translation preserves timed branching bisimulation. ATP is basically an extension of CCS [21] including a timeout operation, an execution delay or *watchdog* operation and the notion of urgent actions. No clocks nor time

variables are considered in ATP. Basically the same study was done by *Daws, Olivero & Yovine* [12] for ET-LOTOS [18]. In this case, also timed branching bisimulation is shown to be preserved. In neither of these works an inverse study was carried out, i.e., to express a timed automata in terms of the process algebra. In particular, it can be shown that ET-LOTOS is less expressive than $\mathcal{T}$, the set of timed automata.

Fokkink [14] sketches an interpretation of ACP with prefix integration [17, 5] into timed automata without invariants. Moreover, the class of strongly regular processes and timed automata turn out to be equivalent when certain restrictions (namely non-Zenoness and fairness) are not present in the behaviour of the timed automata. Thus ACP with prefix integration is more expressive than timed automata. For instance, consider the (finite!) ACP process $\int_{v<1} a[v] \cdot \int_{w=v} b[w] \cdot \mathbf{stop}$, that *records* in v the time when a was performed, and after v units of time executes b. In our language, an unguarded recursive expression would be needed to defined it if the time domain were denumerable. If instead the set of real numbers is considered, such a process cannot be expressed. However, if we allow more expressive constraints by allowing comparison between clocks, we can define $\{\!|x|\!\}\ (x < 1) \boxplus\!\gg a; (\{\!|y|\!\}\ (2y \leq x) \boxplus\!\gg (2y = x) \mapsto b; \mathbf{stop})$. Such an extension would, of course, affect the tractability of the language.

Lynch & Vaandrager [20] introduce a language that explicitly manages clocks. Such a language has the same expressive power as timed automata w.r.t. (weak) timed trace equivalence.

Alur & Henzinger [4] study the extension of programming languages with clock variables. They discuss their semantics in terms of the so called *real-time programs* [15] which are easily translated into timed safety automata (see [15]).

Yi, Pettersson & Daniels [27] give an algebra that represents timed automata without invariants. Basically, the algebra is a syntax for the timed automata including CCS parallel composition and restriction. In particular, the prefixing operation has the form $(\phi, a, C).p$ with $\phi \in \Phi(\mathcal{C})$ and $C \subseteq \mathcal{C}$, and it is the only one that can manage clocks. It could be understood as our term $\phi \mapsto a; \{\!|C|\!\}\ p$. Thus, since terms with conditions in their first actions unavoidably become open terms, it is necessary to consider an initial valuation in its semantics for which $[\mathcal{C} \hookleftarrow 0]$ is taken. That is rather annoying since even when terms like, for instance, $(x < 1, a, \emptyset).\mathbf{stop}$ and $(y < 1, a, \emptyset).\mathbf{stop}$, show the same behaviour, they become different in the context $(\mathbf{tt}, b, \{x\})._{\_}$. Moreover, notice that this language is strictly less expressive than ours, since it does not include invariant operations.

Conclusions. The contribution of this paper is a language for timed automata. This language is basically an extension of Milner's synchronisation trees with operators to handle clocks, namely clocks resettings, invariants and guards. The language has the ability to represent any (image-finite) timed automata by means of guarded recursion, and moreover, any guarded recursive expression can be interpreted as an (image-finite) timed automata. It is extended with the parallel composition and, moreover, some common time operations including time-out, waiting and urgency, are algebraically defined in terms of the basic language.

Also, an equational theory has been given. We have reported that it is sound with respect to timed bisimulation and, moreover, a normal form can be found for each term by using the axioms. With an example we have shown that redundant states, clocks and conditions can be eliminated.

It is interesting to notice that our theory is a conservative extension of Milner's synchronisation trees. We have chosen to use LOTOS-like parallel composition, however, it would also be possible to define the CCS-like parallel composition, restriction and renaming. In such a case, a conservative extension of the CCS calculus could easily be obtained.

In the full version of this paper [10], we introduce a *symbolic bisimulation*, which basically is a bisimulation defined on timed automata. It has the property of implying timed bisimulation, and thus, can simplify proofs of timed bisimilarity. In particular, we use it to prove soundness results. In addition, the full article includes all the proofs omitted here.

Further study includes reachability analysis by using the equational theory, completeness of the axiomatisation, particularly whether it is necessary to include an operator like ACP integration [5], and axiomatisation of other semantic relations as, for instance, timed trace preorder [20].

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In W.R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 1992.
2. 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*, 138:3–34, 1995.
3. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
4. R. Alur and T.A. Henzinger. Real-time system = discrete system + clock variables. In T. Rus and C. Rattray, editors, *Theories and Experiences for Real-Time System Development — Papers presented at First AMAST Workshop on Real-Time System Development,* Iowa City, Iowa, November 1993, pages 1–29. World Scientific, 1994.
5. J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Journal of Formal Aspects of Computing Science*, 3(2):142–188, 1991.
6. J.C.M. Baeten and J.A. Bergstra. Real time process algebra with infinitesimals. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes,* Utrecht, 1994, Workshops in Computing, pages 148–187. Springer-Verlag, 1995.
7. J.C.M. Baeten and J.W. Klop, editors. *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.
8. T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. In P.H.L. van Eijk, C.A. Vissers, and M. Diaz, editors, *The formal description technique LOTOS*, pages 23–73. Elsevier Science Publishers, 1989.
9. T. Bolognesi and F. Lucidi. Timed process algebras with urgent interactions and a unique powerful binary operator. In de Bakker et al. [13], pages 124–148.

10. P.R. D'Argenio and E. Brinksma. A calculus for timed automata. Technical Report CTIT 96-13, Department of Computer Science, University of Twente, 1996.
11. J. Davies et al. Timed CSP: Theory and practice. In de Bakker et al. [13], pages 640–675.
12. C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In Hogrefe and Leue [16], pages 207–222.
13. J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors. *Proceedings REX Workshop on Real-Time: Theory in Practice,* Mook, The Netherlands, June 1991, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.
14. W.J. Fokkink. *Clocks, Trees and Stars in Process Theory.* PhD thesis, University of Amsterdam, December 1994.
15. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.
16. D. Hogrefe and S. Leue, editors. *Proceedings of the 7th International Conference on Formal Description Techniques, FORTE'94.* North-Holland, 1994.
17. A.S. Klusener. *Models and axioms for a fragment of real time process algebra.* PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, December 1993.
18. G. Leduc and L. Léonard. A formal definition of time in LOTOS. In *Revised draft on enhancements to LOTOS*, 1994. Annex G of document ISO/IEC JTC1/SC21/WG1/Q48.6.
19. N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part II: Timing-based systems. Report CS-R9314, CWI, Amsterdam, March 1993. To appear in Information and Computation.
20. N.A. Lynch and F.W. Vaandrager. Action transducers and timed automata. Report CS-R9460, CWI, Amsterdam, November 1994. To appear in *Formal Aspects of Computing.*
21. R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.
22. F. Moller and C. Tofts. A temporal calculus of communicating systems. In Baeten and Klop [7], pages 401–415.
23. X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.
24. X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Transactions on Software Engineering*, 18(9):794–804, September 1992.
25. X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30(2):181–202, 1993.
26. W. Yi. Real-time behaviour of asynchronous agents. In Baeten and Klop [7], pages 502–520.
27. W. Yi, P. Pettersson, and M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In Hogrefe and Leue [16], pages 223–238.

This article was processed using the LaTeX macro package with LLNCS style