Regular Processes and Timed Automata*

Pedro R. D'Argenio

Dept. of Computer Science. University of Twente.

P.O.Box 217. 7500 AE Enschede. The Netherlands.

dargenio@cs.utwente.nl

January 24, 1997

Abstract

In [10], an algebra for timed automata has been introduced. In this article, we introduce a syntactic characterisation of finite timed automata in terms of that process algebra. We show that regular processes, i.e., processes defined using finitely many guarded recursive equations, are as expressive as finite timed automata. The proof uses only the axiom system and unfolding of recursive equations. Since the proofs are basically algorithms, we also provide an effective method to translate from one model into the other. A remarkable corollary of these proofs is that regular recursive specifications may need one clock less than timed automata in order to represent the same process.

1991 Mathematics Subject Classification: 68Q45, 68Q55, 68Q60.

1991 CR Categories: D.3.1, F.3.1, F.3.2, F.4.3.

Keywords: process algebra, real time, timed automata, regular process.

Note: A reduced version of this report was accepted to be published in the *Proceedings of the fourth AMAST Workshop on Real-Time Systems, ARTS'97*, Mallorca, Spain, May 1997.

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

1 Introduction

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, 24, 25, 20, 21, 5, 17, 6, 8, 18]. But the apparently most successful approaches are timed and hybrid automata [3, 22, 15, 2]. Both models have been related in [22, 23, 14, 26, 12]. In [10] both approaches have been integrated and an algebra for timed automata has been introduced. The syntax extends Milner's CCS [19] with operations to manipulate clocks, namely, clock resettings, invariants, and guards. There, an equational theory has been proven to be sound with respect to timed bisimulation. Moreover, the process algebra happens to be as expressive as timed automata. However, in [10] timed automata are not restricted in their amount of locations, edges or clocks.

In this article, we give a syntactic characterisation of *finite* timed automata, that is, timed automata with finitely many locations, edges, and clocks. We show that regular processes, i.e., processes defined using finitely many guarded recursive equations, are as expressive as finite timed automata. This connection is evident between CCS and labelled transition systems where regular processes are defined, either as those processes that are equivalent to a finite labelled transition system, or as those processes that can be rewritten into guarded recursive equations. Anyway, both definitions happen to be equivalent.

Although the relation between our process algebra and the timed automata model is not unexpected, its proof is rather far from being trivial, and moreover, we will find some surprising results in between. Basically, the problems appear because this process algebra has incorporated (clock) variables and the notion of binding. Let us see some examples. In our language we represent the clock resettings by $\{\!|x|\!\}\ p$, the invariants by $\psi \boxright p$ and the guards by $\phi \mapsto p$, where ψ and ϕ are some constraints on the clock variables. We notice that the operation of clock resetting binds clock variables. For some expressions like

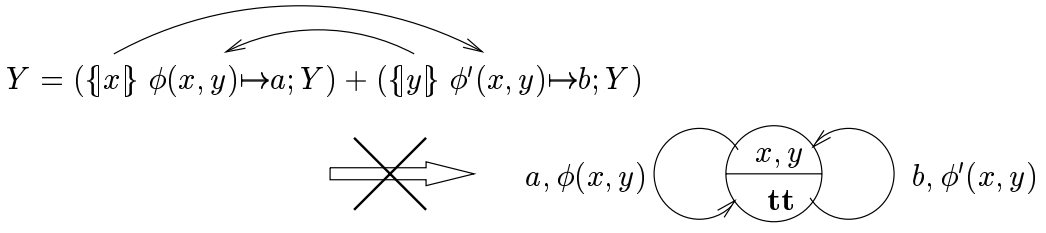
$$Z = \{\!|x|\!\}\ (x \leq 2) \boxright (x \geq 1) \mapsto a; Z \qquad \Longrightarrow \qquad \begin{array}{c} x \\ \hline x \leq 2 \end{array} \quad a, x \geq 1$$

the respective timed automata is straightforwardly obtained. However, some other expressions do not have an obvious associated timed automata. For instance, in

$$X = (x < 3) \boxright \{\!|x|\!\}\ (x < 2) \boxright a; X \qquad \not\Longrightarrow \qquad \begin{array}{c} x \\ \hline x < 3 \\ x < 2 \end{array} \quad a, \mathbf{tt} \tag{1}$$

a naive attempt to associate a timed automaton to X will derive in the one depicted on the right-hand side. However, this is not correct since the x in the invariant $(x < 3)$ is not bound to the clock resetting that follows. This illegal binding is shown by the arrow

on the equation. Even less obvious is the following case, where $\phi(x,y)$ and $\phi'(x,y)$ are some constraints containing clock x and y.



(2)

In this expression, the x on the right-hand side of $+$ should not be bound to the resetting on the left-hand side, and symmetrically for y. Compare with the naive associated timed automaton depicted beside the equation.

In this article, we show that for each finite timed automata there is a regular recursive specification and vice versa. Moreover, we prove this by using only the axiom system and unfolding of recursive equations, which shows the power of the equational theory. It is also important to remark that all the proofs are basically algorithms, thus we are also providing an effective method to translate one model into the other. A remarkable corollary of these proofs is that regular recursive specifications may need one clock less than timed automata in order to represent the same process.

We should point out the related work [4] where a Kleene theorem for timed automata is presented. The language introduced there is shown to be as expressive as timed automata up to timed trace equivalence. Instead, our approach preserves timed bisimilarity. Although our work is perhaps less ambitious, our intention has been to emphasise on the connection of the timed automata theory with the process algebra and its equational theory.

The rest of the article is organised as follows. Section 2 reviews the model of timed automata. In Section 3 we describe the process algebra of [10]. Section 4 is the core of this article and both relations are discussed and proven.

Acknowledgements. I would like to thank Ed Brinksma for his valuable suggestions and discussions. I am grateful to Ahmed Bouajjani and Joseph Sifakis who posed the main question answered in this paper while I was visiting the VERIMAG laboratory in Grenoble, in March 1996.

2 Timed Automata

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:

$$\phi ::= \ \mathbf{tt} \ \mid \ x \Box d \ \mid \ x - y \Box d \ \mid \ (\phi \wedge \phi) \ \mid \ (\neg \phi)$$

where $d \in \mathbb{R}^{\geq 0}$, $\Box \in \{\leq, \geq\}$, and $x, y \in \mathcal{C}$. The abbreviations $\mathbf{ff}$, $x = d$, $x - y < d$, $\phi \vee \phi'$, etc. are defined as usual. Let $var(\phi)$ be the set of clocks occurring in ϕ.

A *(clock) valuation* is a function $v : \mathcal{C} \to \mathbb{R}^{\geq 0}$. Let $\mathcal{V}$ denote the set of valuations. Let $C \subseteq \mathcal{C}$. We define $v[C \leftarrow 0]$ as $v[C \leftarrow 0](x) \stackrel{\text{def}}{=}$ **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, i.e., $var(v(\phi)) = \emptyset$.

Given a valuation v, we define the satisfaction predicate $\models v(\phi)$ as follows

$$\models v(\mathbf{tt}) \qquad \frac{v(x)\Box d}{\models v(x \Box d)} \qquad \frac{v(x) - v(y) \Box d}{\models v(x - y \Box d)} \qquad \frac{\models v(\phi) \quad \models v(\phi')}{\models v(\phi \wedge \phi')} \qquad \frac{\not\models v(\phi)}{\models v(\neg \phi)}$$

We generalise $\models$ to all clock constraints. 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 $\phi \in \overline{\Phi}(\mathcal{C}) \stackrel{\text{def}}{\iff} (\forall v \in \mathcal{V}, d \in \mathbb{R}^{\geq 0}.\ \models (v + d)(\phi) \implies \models v(\phi))$. 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* following the style of [17]

Definition 2.1 Let $\mathbf{A}$ be a set of *actions*. A *timed transition system* (TTS) is a structure $L = (\Sigma, \mathbf{A} \times \mathbb{R}^{\geq 0}, \sigma_0, \longrightarrow, \mathcal{U})$ where

- Σ is a set of *states*, with the *initial state* $\sigma_0 \in \Sigma$;
- $\longrightarrow \subseteq \Sigma \times (\mathbf{A} \times \mathbb{R}^{\geq 0}) \times \Sigma$ is the *transition relation*; and
- $\mathcal{U} \subseteq \mathbb{R}^{\geq 0} \times \Sigma$ is the *until predicate*.

We use the following notation: $a(d)$ iff $(a, d) \in \mathbf{A} \times \mathbb{R}^{\geq 0}$, $\sigma \stackrel{a(d)}{\longrightarrow} \sigma'$ iff $\langle \sigma, a(d), \sigma' \rangle \in \longrightarrow$, $\mathcal{U}_d(\sigma)$ iff $\langle d, \sigma \rangle \in \mathcal{U}$, $\sigma \stackrel{a(d)}{\longrightarrow}$ iff $\exists \sigma' \in \Sigma.\ \sigma \stackrel{a(d)}{\longrightarrow} \sigma'$. In addition, L should satisfy the following axioms:

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

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

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

Definition 2.2 Let $L_i = (\Sigma_i, \mathbf{A} \times \mathbb{R}^{\geq 0}, \sigma_0^i, \longrightarrow_i, \mathcal{U}^i)$, $i \in \{1, 2\}$, be two TTS. A *timed bisimulation* is a relation $R \subseteq \Sigma_1 \times \Sigma_2$ with $\sigma_0^1 R \sigma_0^2$ satisfying, for all $a(d) \in \mathbf{A} \times \mathbb{R}^{\geq 0}$, the following transfer properties:

1. if $\sigma_1 R \sigma_2$ and $\sigma_1 \stackrel{a(d)}{\longrightarrow}_1 \sigma_1'$, then $\exists \sigma_2' \in \Sigma_2.\ \sigma_2 \stackrel{a(d)}{\longrightarrow}_2 \sigma_2'$ and $\sigma_1' R \sigma_2'$;
2. if $\sigma_1 R \sigma_2$ and $\sigma_2 \stackrel{a(d)}{\longrightarrow}_2 \sigma_2'$, then $\exists \sigma_1' \in \Sigma_1.\ \sigma_1 \stackrel{a(d)}{\longrightarrow}_1 \sigma_1'$ and $\sigma_1' R \sigma_2'$; and

3. if $\sigma_1 R \sigma_2$, then $\mathcal{U}_d^1(\sigma_1) \iff \mathcal{U}_d^2(\sigma_2)$.

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

The kind of timed transition systems that we adopted has the same expressive power as the time deterministic timed transition systems in the sense of [25] in which transitions are labelled separatedly with actions and time. Translations from one to the other that preserve bisimulation can be easily found.

Timed Automata. We consider a slight variation of timed safety automata [15] as it was defined in [10]. It is straightforward to check that both approaches have the same expressive power.

Definition 2.3 A *timed (safety) automaton* is a structure $(S, \mathbf{A}, \mathcal{C}, s_0, \longrightarrow, \partial, \kappa)$ where:

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

Moreover, we say that a timed automata is *finite* if it has finitely many locations, clocks and edges. □

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 location s, it can change to be in location s' by performing an action a provided that the clock constraint ϕ holds. The clock resetting function states which clocks should be reset as soon as a location is reached. The invariant assignment function states that the system can idle in location s as long as $\partial(s)$ holds. Formally speaking, a timed automaton can be interpreted as a TTS as follows.

Definition 2.4 Let $T = (S, \mathbf{A}, \mathcal{C}, s_0, \longrightarrow, \partial, \kappa)$ 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 $(\!|T|\!)_{v_0} \stackrel{\text{def}}{=} (S \times \mathcal{V}, \mathbf{A} \times \mathbb{R}^{\geq 0}, (s_0, v_0), \longrightarrow, \mathcal{U})$ where $\longrightarrow$ and $\mathcal{U}$ are defined as the least sets satisfying the following rules:

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

□

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 $\models (v[\kappa(s) \leftarrow 0] + d)(\partial(s))$ 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.

3 A Process Algebra for Timed Automata

In this section we describe the process algebra introduced in [10, 9]. First, we introduce the basic syntax of the process algebra with recursion. Afterwards, we describe the semantics in terms of timed automata. Finally, we introduce the axiom system.

Syntax. The language is an extension of the basic CCS with operations to deal with clocks. We do not introduce the extended language (i.e. parallel composition, hiding operator) in this article since the new operators may be eliminated, that is, any term in the extended language has an equivalent term in the basic language. In addition, the basic language is sufficient for the purpose of this article. The complete language can be found in [9].

Definition 3.1 Let $\mathbf{A}$ be a set of actions, let $\mathcal{C}$ be a set of clocks, and let $\mathbf{V}$ be a set of *process variables*. The language $\mathcal{L}$ is defined according to the following grammar:

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

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

A *recursive specification* E is a set of *recursive equations* having the form $X = p$ for each $X \in \mathbf{V}$, where $p \in \mathcal{L}$. Every recursive specification has a distinguished process variable called *root*. □

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*, can perform 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 often write $\{\!|x_1,\ldots,x_n|\!\}\, p$ instead of $\{\!|\{x_1,\ldots,x_n\}|\!\}\, p$. $\psi \rhd\!\!> p$, the *invariant operation*, can idle while ψ holds or go on with the process p. $p+q$ is the *choice*; it behaves either like p or q. The choice between p and q can be made only by actions, not by the passage of time. The meaning of a variable X depends on its definition in E. Thus, if $X = p \in E$, the behaviour of X is the same as p.

For the sake of completeness, we remark that a recursive specification defines only once each process variable (i.e. $X = p, X = q \in E$ implies $p \equiv q$), and all the variables that occur in any right-hand side of a recursive equation of E are defined in E (i.e. if Y occurs in p and $X = p \in E$, then $Y = q \in E$ for some $q \in \mathcal{L}$.)

Definition 3.2 Let $\nu : \mathbf{V} \to \wp(\mathcal{C})$ be a function that assigns a set of clock variables to each process variable. Define fv^ν inductively as follows,

$$\begin{array}{lcllcl}
fv^\nu(\mathbf{stop}) & = & \emptyset & fv^\nu(\{\!|C|\!\}\, p) & = & fv^\nu(p)\backslash C \\
fv^\nu(a;p) & = & fv^\nu(p) & fv^\nu(\psi \rhd\!\!> p) & = & var(\psi) \cup fv^\nu(p) \\
fv^\nu(\phi\mapsto p) & = & var(\phi) \cup fv^\nu(p) & fv^\nu(X) & = & \nu(X) \\
fv^\nu(p+q) & = & fv^\nu(p) \cup fv^\nu(q) & & &
\end{array}$$

The function $fv : \mathbf{V} \to \wp(\mathcal{C})$, that gives the set of *free variables* of a given process variable, is defined as the least fixed point of the function F defined by

$$F(\nu) = \lambda X.(\nu(X) \cup fv^{\nu}(p))$$

provided that $X = p \in E$. We extend the notion of free variables to every process p by defining $fv(p) \stackrel{\text{def}}{=} fv^{fv}(p)$. □

Now fv can be calculated using the straightforward algorithm to obtain the least fixed point of the monotonic function F. (See Appendix.)

As we already remarked in the introduction, the term $\{\!|C|\!\}\ p$ binds clocks in C which occur free in p. Thus, we have to avoid undesirable bindings and so, we would like to characterise the conflicting terms like (1) or (2). 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 $a; q$. For instance, if $p \equiv (\{\!|x|\!\}\ a; \{\!|y|\!\}\ \phi \mapsto b; X) + (\{\!|z|\!\}\ \psi \rhd\!\!\rhd c; Y)$, then $K_p = \{x, z\}$. 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 $\psi \rhd\!\!\rhd p$) 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 α-conversion) in order to avoid this problem. In [9] we study α-conversion and conflict of variables extensively.

Definition 3.3 An *occurrence of X is guarded* in a term $p \in \mathcal{L}$ if p has a subterm $a; q$ such that this occurrence of X is in q. A *process variable X is guarded* in p if every occurrence of it is guarded. A *term p is guarded* if all its process variables are guarded. A *recursive specification E is strictly guarded* if the righ-hand side of every recursive equation in it is a guarded process, i.e., for all $X = p \in E$, p is guarded. A *recursive specification E is guarded* if it can be rewritten into a strictly guarded recursive specification by properly unfolding the equations. (By "properly unfolding the equations" we mean that whenever $X = p \in E$ and Y occurs unguarded in p, then we take the new specification $E' = (E \backslash \{X = p\}) \cup \{X = p[Y/q]\}$ provided that $Y = q \in E$.)

A recursive specification E is *regular* if it is guarded and defines finitely many recursive equations. □

Proposition 3.4 *Let E be a recursive specification with process variables in $\mathbf{V}$. Define the* unguarded variable dependency graph *G with nodes in $\mathbf{V}$ and edges $X \to Y$ iff Y is unguarded in $X = p \in E$. Then E is guarded if and only if there is no infinite chain in G.*

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

Table 1: Associated timed automata ($X = p \in E$)

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

$$a;p \xrightarrow{a,\mathbf{tt}} p \qquad \frac{p \xrightarrow{a,\phi} p' \quad \partial(p) = \psi}{p+q \xrightarrow{a,\phi\wedge\psi} p' \quad q+p \xrightarrow{a,\phi\wedge\psi} p'} \qquad \frac{p \xrightarrow{a,\phi} p'}{\{\!\{C\}\!\}\ p \xrightarrow{a,\phi} p'} \qquad \frac{p \xrightarrow{a,\phi} p'}{X \xrightarrow{a,\phi} p'}$$

$$\frac{p \xrightarrow{a,\phi'} p'}{\phi \mapsto p \xrightarrow{a,\phi\wedge\phi'} p'} \qquad \frac{p \xrightarrow{a,\phi} p'}{\psi \rhd\!\!> p \xrightarrow{a,\phi} p'}$$

Definition 3.5 Let $p \in \mathcal{L}$ be a process without conflict of variables. The *timed automaton associated to* p is defined by $[\![p]\!] = (\mathcal{L}, \mathbf{A}, \mathcal{C}, p, \longrightarrow, \partial, \kappa)$ where $\longrightarrow$, ∂ and κ are defined inductively by the rules and equations in Table 1. □

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.

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

Proposition 3.6 *Let E be a guarded specification without conflict of variables, then every process variable defined in E has an associated timed automata.*

The proof of Proposition 3.6 follows by induction. In particular, notice that the definitions of κ, ∂ and $\longrightarrow$ are not recursive for the base cases, namely **stop** and prefixing.

We can define the semantics of a process in terms of TTS by first associating a timed automaton and then obtaining the interpretation of it in terms of TTS according to Definition 2.4. Thus, $(\![[\![p]\!]]\!)_{v_0}$ is the *interpretation of p with initial valuation v_0*. Now, we can extend the notion of bisimilarity to processes: two process p and q are *bisimilar* (notation $p \leftrightarrows q$) if for all valuations $v_0 \in \mathcal{V}$, $(\![[\![p]\!]]\!)_{v_0} \leftrightarrows (\![[\![q]\!]]\!)_{v_0}$.

Alternatively, the language has a direct semantics in terms of TTS that is equivalent (modulo bisimulation) to the semantics in two steps given above [10]. We do not present here such semantics since it is not relevant for obtaining the results in this article. Nonetheless, it is essential that the reader understands that this semantics defines a TTS for every guarded recursive specification, including those with conflict of variables. The crucial point of this is that it is not clear whether expressions like (1) and (2) have an associated timed automaton, although they have a clear meaning in terms of TTS.

Axiom system. We introduce a set of axioms for the language described above. The axiom system is sound with respect to bisimulation. Since α-conversion implies bisimulation, we consider terms modulo α-conversion without loss of generality.

Table 2: 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	$\mathbf{ff} \mapsto a; p = \mathbf{stop}$	**A3**	$\phi \mapsto p + \phi' \mapsto p = (\phi \vee \phi') \mapsto p$
A1	$p + q = q + p$	**A3′**	$\psi \boxplus p + \psi' \boxplus p = (\psi \vee \psi') \boxplus p$
A2	$(p + q) + r = p + (q + r)$	**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	$\phi \mapsto (\psi \boxplus p) = \psi \boxplus (\phi \mapsto p)$		
G4	$\phi \mapsto (\{\!\{C\}\!\}\ p) = \{\!\{C\}\!\}\ (\phi \mapsto p)$		if $var(\phi) \cap C = \emptyset$
G5	$\phi \mapsto (p + q) = \phi \mapsto p + \phi \mapsto q$		
I1	$\mathbf{tt} \boxplus p = p$		
I2	$\psi \boxplus (\psi' \boxplus p) = (\psi \wedge \psi') \boxplus p$		
I3	$\psi \boxplus (\{\!\{C\}\!\}\ p) = \{\!\{C\}\!\}\ (\psi \boxplus p)$		if $var(\psi) \cap C = \emptyset$
I4	$\psi \boxplus p + \psi \boxplus q = \psi \boxplus (p + q)$		
I5	$\psi \boxplus (\phi \mapsto a; p) + \psi' \boxplus (\phi' \mapsto b; q) = (\psi \vee \psi') \boxplus ((\psi \wedge \phi) \mapsto a; p + (\psi' \wedge \phi') \mapsto b; q)$		
R1	$\{\!\{C\}\!\}\ p = p$		if $C \cap fv(p) = \emptyset$
R2	$\{\!\{C \cup \{y, x\}\}\!\}\ p = \{\!\{C \cup \{y\}\}\!\}\ [x \leftarrow y]p$		
R3	$\{\!\{C\}\!\}\ \{\!\{C'\}\!\}\ p = \{\!\{C \cup C'\}\!\}\ p$		
R4	$\{\!\{C\}\!\}\ p + \{\!\{C\}\!\}\ q = \{\!\{C\}\!\}\ (p + q)$		
D1	$\phi \mapsto a; (\{\!\{y\}\!\}\ p) = \phi \mapsto a; (\{\!\{y\}\!\}\ (x - y \,\square\, d) \boxplus p)$		if $\models (\phi \Rightarrow (x \,\square\, d))$ and $x \neq y$
D2	$\phi \mapsto a; p = \phi \mapsto a; ((x - y \,\square\, d) \boxplus p)$		if $\models (\phi \Rightarrow (x - y \,\square\, d))$ where $\square \in \{\leq, <, \geq, >, =\}$

Axioms in Table 2 could be explained as follows. Axioms **A1**–**A4** are the extension of the CCS axioms. **A4** needs special care. Since in our model we consider time deadlock, it is not generally the case that **stop** is the neutral element for summation.

However, it is a neutral element only for processes with unbound 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 simplified. 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. Finally, **D1** and **D2** state that the difference between clocks is invariant and thus it could be "transported" along the execution. In particular, **D1** explains how this difference is stated.

The term $[x{\leftarrowtail}y]p$, 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, although for process variables it needs an additional explanation. Given a recursive equation $X = p$, $[x{\leftarrowtail}y]X$ is a 'new' process variable Y defined by the equation $Y = [x{\leftarrowtail}y]p$. Thus, given a recursive specification E, we will usually need to extend it if axiom **R2** is applied.

Theorem 3.7 [9] *The axiom system of Table 2 is sound modulo bisimilarity. That is, for all $p, q \in \mathcal{L}$, if $p = q$ is deduced by means of equational reasoning using α-conversion and axioms in Table 2, then $p \underline{\leftrightarrow} q$.*

4 Regular Processes and Finite Timed Automata

In this section we show that there exists a strong connection between finite timed automata and regular recursive specifications. We show indeed that not only any finite timed automata can be expressed by a regular specification, but also that a regular specification always defines a finite timed automaton up to bisimilarity. But, what is more interesting in this last case is that the axiom system together with unfolding of equations is enough to prove that fact[1].

Although the case of obtaining a regular recursive specification from a finite timed automata seems to be intuitively clear, it is not the same in the case in the other way around due to processes that have conflict of variables. Let us recall the examples of the introduction. The first example was the following recursive equation

$$X = (x < 3) \rhd\!\!> \{\!|x|\!\} \ (x < 2) \rhd\!\!> a; X \tag{1}$$

Notice that the x of $(x < 3)$ should not be bound to the resetting that follows it. Thus, the associated timed automaton is not obvious. Even less obvious is the case of our

[1] Actually, axioms **D1** and **D2** are not necessary.

second example,

$$Y = (\{\!|x|\!\}\ \phi(x,y)\mapsto a; Y) + (\{\!|y|\!\}\ \phi'(x,y)\mapsto b; Y) \tag{2}$$

Notice here, that we can try to calculate $\kappa(Y) = \{x, y\}$, but this is not correct since we may bind some free occurrences of x and y. By the time being, we expect to have motivated the reader and make him/her wonder about the associated timed automata of such equations. We will come back later to these examples.

From finite timed automata to regular recursive expressions. In this paragraph we recall the result already presented in [10]. 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 have a static view and not the usual dynamic notion of reachability in timed automata theory (cf. [1]).

Theorem 4.1 *For every image-finite and finitely sorted timed automaton 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 $[\![X_{s_0}]\!]$ are isomorphic.*

Proof (Sketch). 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 as follows. Let $T = (S, \mathbf{A}, \mathcal{C}, s_0, \longrightarrow, \partial, \kappa)$. For each state $s \in S$ define a different variable X_s. Define the recursive specification E with root X_{s_0} and recursive equations

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

where $\sum\{p_i \mid i \in \{1 \ldots n\}\} \stackrel{\text{def}}{=} p_1 + p_2 + \cdots + p_n$. In particular, $\sum \emptyset \stackrel{\text{def}}{=} \mathbf{stop}$[2].

Now, we restrict T and $[\![X_{s_0}]\!]$ to their reachable parts, and the isomorphism is given by the function that maps every state in T into its corresponding variable in $[\![X_{s_0}]\!]$. □

The following corollary is immediate.

[2]Notice that we are using general summation only when the summands have the form $\phi\mapsto a; p$. If we considered general summation over any kind of processes, then **stop** would not be the best definition for summations over an empty set (this is due to the fact that our model consider time deadlock).

Corollary 4.1.1 *For every finite timed automaton T there exists a regular specification E with root variable X_{s_0} such that T and $[\![X_{s_0}]\!]$ are bisimilar.*

From regular recursive expressions to finite timed automata. In this paragraph, we show how to come up with a timed automaton from a given regular recursive expression. We use the following strategy to construct the timed automata. First, we rewrite the recursive specification into a new one such that each variable, according to the new definition, can execute at most one action and move to another variable. We say that this specification is in *one-step normal form*. The second step translates this last specification into a recursive specification that trivially resembles a timed automaton. We say that this last specification is in *TA-normal form* (TA stands for "Timed Automata").

Definition 4.2 A guarded recursive specification E is in *one-step normal form* if for every $X = p \in E$, p is in the language defined by the following grammar,

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

where $a \in \mathbf{A}$, $\phi \in \Phi(\mathcal{C})$, $\psi \in \overline{\Phi}(\mathcal{C})$, $C \subseteq \mathcal{C}$, and $X \in \mathbf{V}$. □

Lemma 4.3 *Any regular recursive specification can be rewritten into a one-step normal form.*

Proof. The proof follows by adding new process variables, and folding and unfolding when needed. We do that in two steps. First, we reduce the original specification E_0 into a new specification $\overline{E}_0$ such that no prefixing occurs in the scope of another prefixing. That is done by creating new recursive equations. In the second step, $\overline{E}_0$ is taken into one-step normal form by unfolding all the variables that occur unguarded.

Let E_0 be a regular recursive specification. Let $\mathit{size}(p)$ be the number of symbols in p which are not process variables. Choose, if it exists, $X = p \in E_0$ such that p has an occurrence $a; q$ where q is not a process variable and $\mathit{size}(p) \geq \mathit{size}(r)$ for all $Z = r \in E_0$, i.e., p is maximal according to size. Choose a fresh process variable Y and define p' to be p with the occurrence of $a; q$ replaced by $a; Y$. Define

$$E_1 \stackrel{\text{def}}{=} (E_0 \backslash \{X = p\}) \cup \{X = p', Y = q\}$$

Clearly E_1 represents the same process than E_0. Repeat the process until there is no occurrence of $a; q$ with q as before. The algorithm terminates since the function $\langle \mathrm{MAX}(E_i), \#\{X = p \in E_i |\ \mathit{size}(p) = \mathrm{MAX}(E_i)\}\rangle$ strictly decreases in each iteration according to the lexicographical order. We have taken $\mathrm{MAX}(E_i) = \max\{\mathit{size}(p) |\ X = p \in E_i\}$

Let $\overline{E}_0$ be the output of the previous algorithm. Obviously, $\overline{E}_0$ is also a regular specification. Notice that for every equation $X = q \in \overline{E}_0$, by construction, q is in the language defined by the grammar

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

that is, q is "almost" in one-step normal form since it still can have unguarded variables. Construct the unguarded variable dependency graph G_0 according to Proposition 3.4, which implies that G_0 does not have infinite path and hence, it is acyclic. Chose a leaf Y in G_0 with $Y = q \in \overline{E}_0$. For every $X = p \in \overline{E}_0$ such that $X \to Y$ define p' to be p with every unguarded occurrence of Y replaced by q. Define

$$\overline{E}_1 \stackrel{\text{def}}{=} (\overline{E}_0 \setminus \bigcup\{X = p \mid X \to Y\}) \cup \bigcup\{X = p' \mid X \to Y\}$$

Notice that the unguarded dependency graph G_1 of $\overline{E}_1$ is the same as G_0 where all edges $X \to Y$ were removed. So we can repeat the process and the algorithm eventually terminates since the amount of edges is strictly decreasing in each iteration.

It is easy to check that the output of this last algorithm is a recursive specification in one-step normal form. □

Definition 4.4 A recursive specification E is in *TA-normal form* if it is regular and for every $X = p \in E$, p has the form

$$\{\!|x|\!\}\ \psi \rhd \Big(\sum\{\phi_i \mapsto a_i; X_i \mid i \in I\}\Big)$$

where I is a finite index set, $\psi \in \overline{\Phi}(\mathcal{C})$, $x \in \mathcal{C}$, and for all $i \in I$, $a_i \in \mathbf{A}$, $\phi_i \in \Phi(\mathcal{C})$, and $X_i \in \mathbf{V}$. □

Notice that a recursive specification in TA-normal form represents a finite timed automaton. Notice that each variable represents a location and for each of them we have defined only one resetting and one invariant. Moreover, the summation defines the outgoing edges labelled with the respective guard and action. (See the proof of Theorem 4.1.)

Theorem 4.5 *Any regular specification can be rewritten into a TA-normal form by using foldings, unfoldings, renaming of clock variables and axioms in Table 2.*

Proof. Let E be a regular specification with root X_0. Because of Lemma 4.3, we may assume that E is already in one-step normal form. Let E' be the recursive specification defined as follows. For every process variable X defined in E with free variables $fv(X) = \{x_1, \ldots, x_n\}$, define a process variable $X_{x_1,\ldots,x_n}$. Obviously, such a relation is a bijection. Now, for every $X = p \in E$ define $X_{x_1,\ldots,x_n} = p' \in E'$ where p' is the same as p but with all process variables replaced by their respective images. Define the root of E' as the image of X_0. So, E' is the same as E with the names of the process variables changed.

We will need to consider renaming of clock variables. We do that in the expected way, except that in the case of process variables, instead of propagating the renaming in the definition of the variable, we will just rearrange the subindex. Thus,

$$[x_i \leftarrow y] X_{x_1,\ldots,x_i,\ldots,x_n} \stackrel{\text{def}}{=} X_{x_1,\ldots,y,\ldots,x_n} \tag{3}$$

In such a case, we need to consider an extended set of process variables. Let $\mathcal{C}$ be the set of clocks occurring in E (or similarly, in E'). Let $\overline{x} \notin \mathcal{C}$ be a new clock. Let $\mathbf{V}$ be the original set of process variables defined in E. We define

$$\mathbf{V}' \stackrel{\text{def}}{=} \{X_{x_1,\ldots,x_n} \mid X \in \mathbf{V} \wedge n = \#fv(X) \wedge \forall i \in \{1,\ldots,n\}.\, x_i \in \mathcal{C} \cup \{\overline{x}\}\}$$

From now on, we will denote with $\overline{X}, \overline{X}_i, \ldots$ the variables in $\mathbf{V}'$. The necessity of the new clock $\overline{x}$ will be clear in the following calculations, since sometimes we will need to chose a fresh clock variable.

For each $X_{x_1,\ldots,x_n} = p \in E'$ we proceed by induction according to Definition 4.2. More precisely, we will show that p has the form

$$\{\!|\overline{x}|\!\}\ \psi \rhd\!\!> \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i$$

(Here $\sum_{i\in I} p_i$ abbreviates $\sum\{p_i \mid i \in I\}$.)

Case **stop**.

$$\mathbf{stop} \stackrel{\mathbf{R1},\mathbf{I1}}{=} \{\!|\overline{x}|\!\}\ \mathbf{tt} \rhd\!\!> \mathbf{stop}$$

Case $a; \overline{X}$.

$$a;\overline{X} \stackrel{\mathbf{G1}}{=} \mathbf{tt}\mapsto a;\overline{X} \stackrel{\mathbf{R1},\mathbf{I1}}{=} \{\!|\overline{x}|\!\}\ \mathbf{tt} \rhd\!\!> (\mathbf{tt}\mapsto a;\overline{X})$$

Case $\phi \mapsto p$. By induction hypothesis assume $p = \{\!|\overline{x}|\!\}\ \psi \rhd\!\!> \sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i$. Since $\overline{x}$ is fresh, $\overline{x} \notin var(\phi)$. If $I \neq \emptyset$ we proceed as follows.

$$\phi\mapsto p \stackrel{\mathbf{G4},\mathbf{G3}}{=} \{\!|\overline{x}|\!\}\ \psi \rhd\!\!> \phi \mapsto \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i \stackrel{\mathbf{G5},\mathbf{G2}}{=} \{\!|\overline{x}|\!\}\ \psi \rhd\!\!> \sum_{i\in I} \phi \wedge \phi_i \mapsto a_i; \overline{X}_i$$

If $I = \emptyset$, then

$$\phi\mapsto p \stackrel{\mathbf{G4},\mathbf{G3}}{=} \{\!|\overline{x}|\!\}\ \psi \rhd\!\!> \phi \mapsto \mathbf{stop} \stackrel{\mathbf{G0}}{=} \{\!|\overline{x}|\!\}\ \psi \rhd\!\!> \mathbf{stop}$$

Case $p + q$. By induction hypothesis assume $p = \{\!|\overline{x}|\!\}\ \psi \rhd\!\!> \sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i$ and $q = \{\!|\overline{x}|\!\}\ \psi' \rhd\!\!> \sum_{j\in J} \phi'_j \mapsto b_j; \overline{Y}_j$. If $I \neq \emptyset$ and $J \neq \emptyset$, then

$$\begin{aligned}
p + q &\stackrel{\text{IH}}{=} \left(\{\!|\overline{x}|\!\}\ \psi \rhd\!\!> \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i\right) + \left(\{\!|\overline{x}|\!\}\ \psi' \rhd\!\!> \sum_{j\in J} \phi'_j \mapsto b_j; \overline{Y}_j\right) \\
&\stackrel{\mathbf{R4},\mathbf{I4}}{=} \{\!|\overline{x}|\!\}\ \left(\left(\textstyle\sum_{i\in I} \psi \rhd\!\!> \phi_i \mapsto a_i; \overline{X}_i\right) + \left(\sum_{j\in J} \psi' \rhd\!\!> \phi'_j \mapsto b_j; \overline{Y}_j\right)\right) \\
&\stackrel{\mathbf{A1},\mathbf{A2},\mathbf{A3}}{=} \{\!|\overline{x}|\!\}\ \left(\left(\textstyle\sum_{i\in I}(\psi \rhd\!\!> \phi_i \mapsto a_i; \overline{X}_i) + (\psi' \rhd\!\!> \phi'_1 \mapsto b_j; \overline{Y}_1)\right)\right. \\
&\qquad \left. + \left(\textstyle\sum_{j\in J}(\psi' \rhd\!\!> \phi'_j \mapsto b_j; \overline{Y}_j) + (\psi \rhd\!\!> \phi_1 \mapsto a_i; \overline{X}_1)\right)\right) \\
&\stackrel{\mathbf{I5}}{=} \{\!|\overline{x}|\!\}\ \left(\left(\textstyle\sum_{i\in I}(\psi \vee \psi') \rhd\!\!> ((\psi \wedge \phi_i) \mapsto a_i; \overline{X}_i + (\psi' \wedge \phi'_1) \mapsto b_j; \overline{Y}_1)\right)\right. \\
&\qquad \left. + \left(\textstyle\sum_{j\in J}(\psi \vee \psi') \rhd\!\!> ((\phi'_j \wedge \psi') \mapsto b_j; \overline{Y}_j + (\psi \wedge \phi_1) \mapsto a_i; \overline{X}_1)\right)\right) \\
&\stackrel{\mathbf{I4}}{=} \{\!|\overline{x}|\!\}\ (\psi \vee \psi') \rhd\!\!> \left(\left(\textstyle\sum_{i\in I}(\psi \wedge \phi_i) \mapsto a_i; \overline{X}_i + (\psi' \wedge \phi'_1) \mapsto b_j; \overline{Y}_1\right)\right. \\
&\qquad\qquad \left. + \left(\textstyle\sum_{j\in J}(\phi'_j \wedge \psi') \mapsto b_j; \overline{Y}_j + (\psi \wedge \phi_1) \mapsto a_i; \overline{X}_1\right)\right) \\
&\stackrel{\mathbf{A1},\mathbf{A2},\mathbf{A3}}{=} \{\!|\overline{x}|\!\}\ (\psi \vee \psi') \rhd\!\!> \left(\left(\textstyle\sum_{i\in I}(\psi \wedge \phi_i) \mapsto a_i; \overline{X}_i\right)\right. \\
&\qquad\qquad \left. + \left(\textstyle\sum_{j\in J}(\phi'_j \wedge \psi') \mapsto b_j; \overline{Y}_j\right)\right)
\end{aligned}$$

If $I = \emptyset$ and $J \neq \emptyset$, then, for some $a \in \mathbf{A}$ and $\overline{Z} \in \mathbf{V}'$, we have

$$\begin{array}{rcl} p + q & \overset{\text{IH}}{=} & (\{\!|\overline{x}|\!\}\ \psi \mathbin{|\!\gg} \mathbf{stop}) + \left(\{\!|\overline{x}|\!\}\ \psi' \mathbin{|\!\gg} \textstyle\sum_{j\in J} \phi'_j \mapsto b_j; \overline{Y}_j\right) \\ & \overset{\mathbf{Stp}}{=} & \left(\{\!|\overline{x}|\!\}\ \psi \mathbin{|\!\gg} \mathbf{ff} \mapsto a; \overline{Z}\right) + \left(\{\!|\overline{x}|\!\}\ \psi' \mathbin{|\!\gg} \textstyle\sum_{j\in J} \phi'_j \mapsto b_j; \overline{Y}_j\right) \end{array}$$

Now, we follows as before and we obtain

$$\begin{array}{rcl} & = & \{\!|\overline{x}|\!\}\ (\psi \vee \psi') \mathbin{|\!\gg} \left(\left(\mathbf{ff} \mapsto a; \overline{Z}\right) + \left(\textstyle\sum_{j\in J} (\phi'_j \wedge \psi') \mapsto b_j; \overline{Y}_j\right)\right) \\ & \overset{\mathbf{Stp,A4}}{=} & \{\!|\overline{x}|\!\}\ (\psi \vee \psi') \mathbin{|\!\gg} \left(\textstyle\sum_{j\in J} (\phi'_j \wedge \psi') \mapsto b_j; \overline{Y}_j\right) \end{array}$$

The other cases are similar.

Case $\{\!|C|\!\}\ p$. By induction hypothesis assume $p = \{\!|\overline{x}|\!\}\ \psi \mathbin{|\!\gg} \sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i$. Then

$$\begin{array}{rcl} \{\!|C|\!\}\ p & \overset{\text{IH}}{=} & \{\!|C|\!\}\ \{\!|\overline{x}|\!\}\ \psi \mathbin{|\!\gg} \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i \\ & \overset{\mathbf{R3}}{=} & \{\!|C \cup \{\overline{x}\}|\!\}\ \psi \mathbin{|\!\gg} \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i \\ & \overset{\mathbf{R2}}{=} & \{\!|\overline{x}|\!\}\ ([C \hookleftarrow \overline{x}]\psi) \mathbin{|\!\gg} \textstyle\sum_{i\in I} ([C \hookleftarrow \overline{x}]\phi_i) \mapsto a_i; [C \hookleftarrow \overline{x}]\overline{X}_i \end{array}$$

Case $\psi \mathbin{|\!\gg} p$. By induction hypothesis assume $p = \{\!|\overline{x}|\!\}\ \psi' \mathbin{|\!\gg} \sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i$. Since $\overline{x}$ is fresh, $\overline{x} \notin \mathit{var}(\psi)$. Then

$$\psi \mathbin{|\!\gg} p \overset{\text{IH}}{=} \psi \mathbin{|\!\gg} \{\!|\overline{x}|\!\}\ \psi' \mathbin{|\!\gg} \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i \overset{\mathbf{I3,I2}}{=} \{\!|\overline{x}|\!\}\ (\psi \wedge \psi') \mathbin{|\!\gg} \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i$$

Now, the obtained recursive equations are in TA-normal form, i.e.,

$$X_{x_1,\ldots,x_n} = \{\!|\overline{x}|\!\}\ \psi' \mathbin{|\!\gg} \sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i \tag{4}$$

However, we have been a bit sloppy since now we have a lot of process variables which have not been defined. They come from *Case* $\{\!|C|\!\}\ p$ when we had to rename clock variables by applying axiom **R2**. We are going to define those variables.

Suppose that the variable $X_{y_1,\ldots,y_n}$ is undefined. We define it as the appropriate renaming of $X_{x_1,\ldots,x_n}$ already defined in (4). First, notice that for each equation like (4) we need $n + 1$ clock variables: n is the amount of free variables and the other one is the clock resetting of $\overline{x}$. Now, suppose that some of $y_1, \ldots, y_n$ are $\overline{x}$, so there must be a clock $\overline{y}$ not in $\{y_1, \ldots, y_n\}$. If none of $y_1, \ldots, y_n$ is $\overline{x}$, then we take $\overline{y} = \overline{x}$. Now, $X_{y_1,\ldots,y_n}$ is defined as follows

$$X_{y_1,\ldots,y_n} = \{\!|\overline{y}|\!\}\ \ ([\mathcal{X} \hookleftarrow \mathcal{Y}]\psi') \mathbin{|\!\gg}\ \textstyle\sum_{i\in I} ([\mathcal{X} \hookleftarrow \mathcal{Y}]\phi_i) \mapsto a_i; [\mathcal{X} \hookleftarrow \mathcal{Y}]\overline{X}_i \tag{5}$$

where $[\mathcal{X} \hookleftarrow \mathcal{Y}]$ is the simultaneous substitution $[x_1 \hookleftarrow y_1, \ldots, x_n \hookleftarrow y_n, \overline{x} \hookleftarrow \overline{y}]$. This last recursive specification that extends E' by defining all variables in $\mathbf{V}'$ and has as a root variable the image of X^0, is equivalent to E and is in TA-normal form. □

As an example of the proof of Theorem 4.5, we will translate the recursive equation (2) into TA-normal form. First we define the new equation

$$Y_{x,y} = (\{\!|x|\!\}\ \phi(x,y) \mapsto a; Y_{x,y}) + (\{\!|y|\!\}\ \phi'(x,y) \mapsto b; Y_{x,y})$$

according to the first part of the proof. We consider a new clock z and hence $\mathbf{V}' = \{Y_{u,v} \mid u, v \in \{x, y, z\}\}$. Following the inductive deduction, we calculate

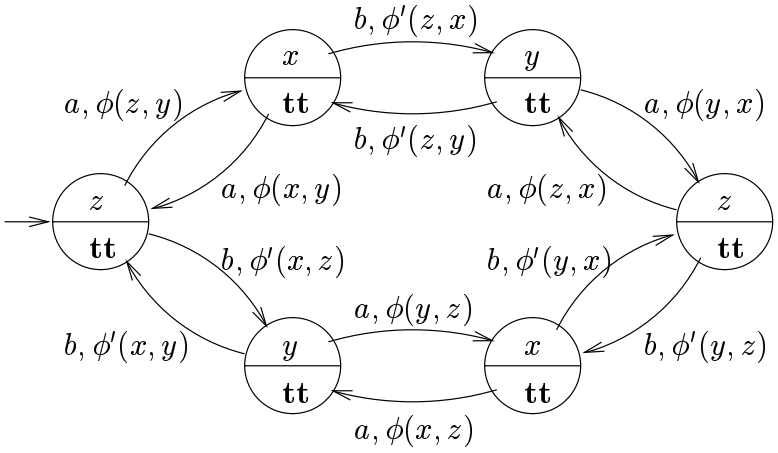
$$\begin{aligned}
Y_{x,y} &= (\{\!|x|\!\}\ \phi(x,y) \mapsto (\{\!|z|\!\}\ \mathbf{tt} \rhd\!\!> \mathbf{tt} \mapsto a; Y_{x,y})) \\
&\quad + (\{\!|y|\!\}\ \phi'(x,y) \mapsto (\{\!|z|\!\}\ \mathbf{tt} \rhd\!\!> \mathbf{tt} \mapsto b; Y_{x,y})) \\
&= (\{\!|x|\!\}\ (\{\!|z|\!\}\ \mathbf{tt} \rhd\!\!> \phi(x,y) \mapsto a; Y_{x,y})) + (\{\!|y|\!\}\ (\{\!|z|\!\}\ \mathbf{tt} \rhd\!\!> \phi'(x,y) \mapsto b; Y_{x,y})) \\
&= (\{\!|z|\!\}\ \mathbf{tt} \rhd\!\!> \phi(z,y) \mapsto a; Y_{z,y}) + (\{\!|z|\!\}\ \mathbf{tt} \rhd\!\!> \phi'(x,z) \mapsto b; Y_{x,z})) \\
&= \{\!|z|\!\}\ \mathbf{tt} \rhd\!\!> (\phi(z,y) \mapsto a; Y_{z,y} + \phi'(x,z) \mapsto b; Y_{x,z})
\end{aligned}$$

Now, according to (5), we can calculate, for instance

$$Y_{z,x} = \{\!|y|\!\}\ \mathbf{tt} \rhd\!\!> (\phi(y,x) \mapsto a; Y_{y,x} + \phi'(z,y) \mapsto b; Y_{z,y})$$

Notice that, in this case, variables $Y_{x,x}$, $Y_{y,y}$, and $Y_{z,z}$ are redundant, i.e., never reached from the root. At this point the reader should not find difficulties to check that the associated timed automaton is the one depicted in Figure 1. Locations are represented by circles. κ and ∂ are respectively written in the upper and lower part of the circle, and edges are represented by the arrows.

Figure 1: $Y = (\{\!|x|\!\}\ \phi(x,y) \mapsto a; Y) + (\{\!|y|\!\}\ \phi'(x,y) \mapsto b; Y)$



Since any regular specification can be rewritten into TA-normal form according to Theorem 4.5, and any recursive specification in TA-normal form defines trivially a finite timed automata, as it was already observed, we obtain the following corollary.

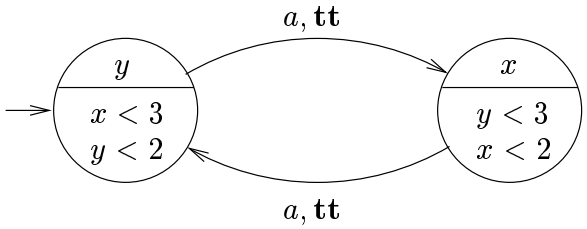
Corollary 4.5.1 *Any regular recursive specification can be proven bisimilar to a finite timed automaton.*

As a second corollary we have that the same expressive power of the timed automata is preserved if we restrict to those automata that reset only one clock in each location, i.e., $\#\kappa(s) \leq 1$. So, by Corollary 4.1.1, Theorem 4.5 and the observation after Definition 4.4, we have:

Corollary 4.5.2 *Every finite timed automaton T is bisimilar to some finite timed automaton T' such that at most one clock is reset in every location of T'.*

A detail to remark is that regular specifications may need fewer clocks than finite timed automata in order to represent the same process. A clear example showing that fact is the expression (1) given above. The corresponding timed automaton is depicted in Figure 2. Notice that both clocks x and y are necessary.

Figure 2: $X = (x < 3) \rhd\!\!> \{\!|x|\!\} \ (x < 2) \rhd\!\!> a; X$



However, we have the interesting result that, for going from a regular specification to a finite timed automaton, we need to add at most one new clock. This follows from the observation that in Lemma 4.3 we do not modify the set $\mathcal{C}$ and to prove Theorem 4.5 we only require to consider only one extra clock. Thus, we have proved the following.

Corollary 4.5.3 *Let E be a regular recursive specification with clocks in $\mathcal{C}$. Then, there exists a finite timed automaton T with clocks in $\mathcal{C}'$ such that T can be proven bisimilar to E and $\#\mathcal{C}' \leq 1 + \#\mathcal{C}$*

Appendix: Calculating *fv*

We have claimed that the proofs in Section 4 are algorithms. In particular, the proof of Theorem 4.5 needs to know which is the set of free variables of a given process variable. Thus, for the sake of completeness, we give an algorithm to calculate the free variables of a process variable.

The algorithm terminates since the function $\sum_{i \in I} \#\nu(X_i)$ strictly increases in each iteration and it is bounded by the amount of clock variables times the amount of process variables.

Figure 3: Algorithm for computing fv

input $\{X_i = p_i \mid i \in I\}$
output $fv := \nu$

for all $i \in I$ **do**
    $\nu(X_i) := \emptyset$
od
repeat
    for all $i \in I$ **do**
        $prev(X_i) := \nu(X_i)$
        $\nu(X_i) := prev(X_i) \cup fv^{\nu}(p_i)$
    od
until $\bigwedge_{i \in I} prev(X_i) = \nu(X_i)$

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] E. Asarin, P. Caspi, and O. Maler. A Kleene theorem for timed automata, April 1996. Unpublished Manuscript.

[5] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 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 F. Lucidi. Timed process algebras with urgent interactions and a unique powerful binary operator. In de Bakker et al. [13], pages 124–148.

[9] 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.

[10] P.R. D'Argenio and E. Brinksma. A calculus for timed automata (Extended abstract). In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems,* Uppsala, Sweden, volume 1135 of *Lecture Notes in Computer Science*, pages 110–129. Springer-Verlag, 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 7^{th} 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] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[20] F. Moller and C. Tofts. A temporal calculus of communicating systems. In Baeten and Klop [7], pages 401–415.

[21] X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.

[22] 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.

[23] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30(2):181–202, 1993.

[24] W. Yi. Real-time behaviour of asynchronous agents. In Baeten and Klop [7], pages 502–520.

[25] W. Yi. CCS + Time = an interleaving model for real time systems. In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Proceedings* 18^{th} *ICALP,* Madrid, volume 510 of *Lecture Notes in Computer Science*, pages 217–228. Springer-Verlag, 1991.

[26] 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.