Controlled Timed Automata

François Demichelis and Wiesław Zielonka

LaBRI*, Université Bordeaux 1,
351 cours de la Libération, 33405 Talence Cedex, France
e-mail: {demichel|zielonka}@labri.u-bordeaux.fr

Abstract. We examine some extensions of the basic model, due to Alur and Dill, of real-time automata (RTA). Our model, controlled real-time automata, is a parameterized family of real-time automata with some additional features like clock stopping, variable clock velocities and periodic tests. We illustrate the power of controlled automata by presenting some languages that can be recognized deterministically by such automata, but cannot be recognized non-deterministically by any other previously introduced class of timed automata (even with ε-transitions). On the other hand, due to carefully chosen restrictions, controlled automata conserve basic properties of RTA: the emptiness problem is decidable and for each fixed parameter the family of recognized real-time languages is closed under boolean operations.

1 Introduction.

The current research on real-time systems, their specification and verification, develops in various directions: real-time automata of various types [2, 4, 3] and much more powerful hybrid systems [8], logics [12, 1], ...

In this paper we concentrate our attention on real-time automata. Our aim is to study possible extensions of the classical model of real-time automata (RTA) of [2]. However, not all possible extensions attract our interest, we want to preserve the main properties of RTA which make this class so attractive.

There are essentially three positive results in [2]:

(P1) For each real-time automaton $\mathcal{A} \in$ RTA the language untime($\mathcal{L}(\mathcal{A})$) obtained by dropping out the time components of all timed words is recognizable and there is an effective procedure allowing to obtain a finite automaton recognizing untime($\mathcal{L}(\mathcal{A})$).

(P2) The family of real-time languages recognized by deterministic real-time automata forms a boolean algebra.

(P3) If $\mathcal{A}, \mathcal{B} \in$ RTA (not necessarily deterministic) then we can construct effectively real-time automaton recognizing $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$ and $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$.

Especially the first two results are important for practical applications of RTA.

* Unité associé au CNRS, UMR 5800.

(P1) implies the decidability of the emptiness problem for RTA — namely it suffices to verify whether the regular language untime($\mathcal{L}(\mathcal{A})$) is empty or not (in this context we can recall also that Alur and Dill [2] show that the universality problem for RTA is not decidable.)

The importance of (P1) and (P2) stems from the fact that these are precisely the properties that allow to use real-time automata as an effective tool for real-time system verification: if both a system Sys and its specification Spec are described by deterministic RTA then (P1)-(P2) allow to check if $\mathcal{L}(\text{Sys}) \subseteq \mathcal{L}(\text{Spec})$. To this end we construct first an automaton $\overline{\text{Spec}}$ recognizing the complement of $\mathcal{L}(\text{Spec})$ and next an automaton $\mathcal{A}$ recognizing $\mathcal{L}(\text{Sys}) \cap \mathcal{L}(\overline{\text{Spec}})$. The emptiness of $\mathcal{L}(\mathcal{A})$ is equivalent with the initial inclusion problem.[1] Although these steps are well-known and understood we recalled them to emphasize again the crucial role of (P1) and (P2) in the verification process.

The aim of our paper is to examine how we can extend RTA but retain properties (P1)-(P2). Such attempts are not new. In [4] and [3] two extensions of RTA are proposed, both of them satisfying (P1) and (P2) and increasing the expressive power of deterministic timed automata. Both models however have the property that the current event can depend not only on the past events but also on the future, in [4] the automaton head can go back and forth scanning a real-time word written on a tape, in [3] there are backward going clocks, their values can be reset by future "not yet occurred" events and observed by the current action.

In our paper we limit our investigations to models where event occurrence is uniquely conditioned by the past. Recently, it was noted by Choffrut and Goldwurm [5] that simply by adding periodic constraints we already increase the power of deterministic RTA. Since [5] shows that periodic constraints can be simulated by RTA with ε-transitions, condition (P1) holds for such automata. The verification of (P2) is also straightforward.

In our paper we define a new class of timed automata: controlled real-time automata (CRTA). Such automata are composed of two parts: a frame *Frm* and a control *Ctr*. In fact CRTA is not one class of automata but rather a family of classes parameterized by *Frm*. Fixing the frame we get a class CRTA_{Frm} (in particular automata with periodic constraints of [5] correspond to the class CRTA_{Frm} with the trivial frame). Our main result is that for any fixed deterministic frame *Frm* the class CRTA_{Frm} satisfies conditions (P1)–(P2). We show also some examples that illustrate how, with some simple but non-trivial frames, we can recognize deterministically languages that are not recognizable even by the most powerful up to now class of non-deterministic timed automata with ε transitions (note that the previous extensions of [4, 3, 5] could be simulated by such automata).

As it turns out controlled real-time automata are in some sense at the limit of what can be done in this direction. Although each class CRTA_{Frm} satisfies (P1)–(P3) separately it does not exists a class of automata encompassing all classes CRTA_{Frm} and still satisfying (P1).

[1] Property (P3) allows also to choose Sys automaton non-deterministic.

2 Preliminaries.

In the sequel $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{N}$ denote the sets of real, rational and non-negative integer numbers respectively. We set also $\mathbb{R}_{\zeta 0} = \{x \in \mathbb{R} \mid x \zeta 0\}$ for $\zeta \in \{>, \geq\}$ (a similar notation will also be used for subsets of $\mathbb{Q}$). Since we use frequently formulas of the form $(a \leq b)$ and $(a \geq b)$ to save place we assume that $\#$ denotes always any of the relations $\{\leq, \geq\}$. For $a \in \mathbb{R}$, $\lfloor a \rfloor$, $\lceil a \rceil$ are respectively the greatest and the smallest integer such that $\lfloor a \rfloor \leq a \leq \lceil a \rceil$, while $\mathrm{fract}(a) = a - \lfloor a \rfloor$ is the fractional part of a.

For any sets A and B, $[\![A \to B]\!]$ denotes the set of all functions from A to B.

For an alphabet Σ, Σ^* and Σ^ω are the sets of finite and infinite words, $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$.

A finite *timed word* (of length n) over an alphabet Σ is a sequence $(t_1, a_1, t_2, a_2, \ldots, t_n, a_n)$ alternating non-negative real numbers $t_i \in \mathbb{R}_{\geq 0}$ and letters of Σ, $a_i \in \Sigma$. An infinite timed word $w = (t_1, a_1, t_2, a_2, \ldots)$ is a similar infinite sequence such that the series $\sum_{i \geq 0} t_i$ is divergent. Intuitively, a_i represent actions executed by a real time system and t_i is the time elapsed between a_{i-1} and a_i.

Sometimes it will be more convenient to use an equivalent notion of *timed occurrence words*. Such a word is a sequence $w = (a_1, \tau_1), (a_2, \tau_2), \ldots$ of pairs $(a_i, \tau_i) \in \Sigma \times \mathbb{R}$ such that $0 \leq \tau_1 \leq \tau_2 \leq \ldots$ and whenever w is infinite then $\lim_{i \to \infty} \tau_i = \infty$. In the timed occurrence word w the pair (a_i, τ_i) indicates an action a_i executed at the moment τ_i. There is an obvious natural bijection between timed words and timed occurrence words: an occurrence word $w = (a_1, \tau_1), (a_2, \tau_2), \ldots$ corresponds to the timed word $w' = (t_1, a_1, t_2, a_2, \ldots)$ such that $\forall n, \tau_n = \sum_{i=1}^{n} t_i$ (or equivalently, $t_1 = \tau_1$ and $\forall n > 1, t_n = \tau_n - \tau_{n-1}$). In the sequel, we switch freely back and forth between timed words and the corresponding timed occurrence words, in particular the definitions expressed in terms of timed words are tacitly extended to the corresponding timed occurrence words and vice versa.

For any timed word $w = (t_1, a_1, t_2, a_2, \ldots)$ by $\mathrm{untime}(w)$ we shall note the word $a_1 a_2 \ldots$ of Σ^∞.

3 Controlled Real-Time Automata.

Controlled timed automata, CRTA for short, are composed of two parts, a frame which is common for all automata of a given class and a control which is specific to a given automaton. To assemble the frame and the control in order to obtain an automaton it is necessary that they have a common signature.

A *signature* is a pair $\sigma = (\Sigma, \Omega)$ consisting of an alphabet Σ and a finite set Ω of *colours*. A *frame* automaton over signature (Σ, Ω) is a tuple

$$Frm = (\Sigma, \Omega, S, s_0, \Delta_{Frm}, \omega_S, \mathrm{vel}) \tag{1}$$

where S is a finite set of states, $s_0 \in S$ is the initial state, $\Delta_{Frm} \subseteq S \times \Sigma \times S$ is a transition relation and finally ω_S and vel are two output mappings $\omega_S : S \to \Omega$, vel : $S \to \mathbb{Q}$.

In other words, *Frm* is a finite transition system that outputs for each state a pair $(\omega_S(s), \text{vel}(s)) \in \Omega \times \mathbb{Q}$.

A finite *control* automaton over signature (Σ, Ω) is a tuple

$$Ctr = (\Sigma, \Omega, X, \omega_X, \mathbf{lower}, \mathbf{upper}, Q, q_0, \Delta_{Ctr}, Q_F, \mathcal{F}) \tag{2}$$

where X is a finite set of clocks, Q is a finite set of states, $q_0 \in Q$ is the initial state,

$$\Delta_{Ctr} = Q \times \Sigma \times \mathbf{Tests} \times \mathbf{Operations} \times Q$$

is a finite transition relation, $Q_F \subseteq Q$ is the set of final states, $\mathcal{F} \subseteq 2^Q$ is a family of subsets of Q, $\omega_X : X \longrightarrow \Omega$ is a clock colouring mapping and finally $\mathbf{upper}, \mathbf{lower} : X \longrightarrow \mathbb{Q}$ are two mappings such that for each clock $x \in X$, $\mathbf{lower}(x) \leq 0 \leq \mathbf{upper}(x)$. The values $\mathbf{lower}(x)$ and $\mathbf{upper}(x)$ are called respectively the lower and the upper boundary of x.

We assume that the values of clock variables range over $\mathbb{R}$ and mappings $v \in [\![X \to \mathbb{R}]\!]$ are called *clock valuations*.

To complete the definition of *Ctr* we should specify the set **Tests** consisting of conditions interpreted over clock valuations and the set **Operations** of operations over clocks.

We begin with the set of elementary tests which consists of two types of formulas:

(1) $(x \# a)$, where $x \in X$ and $a \in \mathbb{Q}$, subject to the condition $\mathbf{lower}(x) \leq a \leq \mathbf{upper}(x)$,
(2) $(x \bmod k) \# b$, for $x \in X$, $k \in \mathbb{N}$, $b \in \mathbb{Q}$, subject to the condition $\mathbf{lower}(x) \leq b \leq \mathbf{upper}(x)$.

The set **Tests** of all tests is defined as the smallest set containing all elementary tests and such that $\varphi_1, \varphi_2 \in \mathbf{Tests}$ implies $\neg\varphi_1, \varphi_1 \wedge \varphi_2 \in \mathbf{Tests}$ (other usual propositional operators are also used for the sake of convenience.) The satisfiability relation $v \models \varphi$, $v \in [\![X \to \mathbb{R}]\!]$, $\varphi \in \mathbf{Tests}$, is defined in the obvious way for elementary tests: $v \models (x \bmod k) \# b$ iff $(v(x) \bmod k) \# b$ and $v \models (x \# a)$ iff $v(x) \# a$ and extends to all tests as usual.

To define the set **Operations** we begin with atomic operations over clocks:

(1) $(x \leftarrow 0)$ — that sets the value of the clock x to 0 (reset operation),
(2) $x \leftarrow (x \bmod k)$, where $k \in \mathbb{N}$ is such that $\mathbf{lower}(x) \leq k \leq \mathbf{upper}(x)$, is an operation changing the valuation of x from $v(x)$ to $(v(x) \bmod k)$ (mod operation).

An operation op $\in$ **Operations** is just a set of elementary operations such that op contains at most one elementary operation for each clock x.

In the sequel, to simplify the notation, the elementary operation $(x \leftarrow 0)$ will be noted as x, while $(x \leftarrow x \bmod k)$ will be noted as $(x \bmod k)$. Therefore, $\{x, y \bmod 3\}$ is an operation resetting x and changing the value of y to $(y \bmod 3)$, while $\{x \bmod 4, x \bmod 6, y\}$ is not a valid operation since it contains two elementary operations over x.

Let $\mathrm{op} \in \mathbf{Operations}$ and $v \in [\![X \to \mathbb{R}]\!]$. The clock valuation resulting in application of op to v will be denoted by $\langle v; \mathrm{op}\rangle$ and is defined in the following way:[2]

$$\langle v; \mathrm{op}\rangle(x) = \begin{cases} 0 & \text{if } \ x \in \mathrm{op} \\ v(x) \bmod k & \text{if } \ (x \bmod k) \in \mathrm{op} \\ v(x) & \text{if op does not contain any elementary operation} \\ & \text{over } x\ . \end{cases}$$

A *controlled timed automaton* is a pair $\mathcal{A} = (\mathit{Frm}, \mathit{Ctr})$ consisting of a frame and a control, both with the same signature. For a fixed frame *Frm* the corresponding class of controlled timed automata is denoted $\mathrm{CRTA}_{\mathit{Frm}}$.

Intuitively, such automaton $\mathcal{A}$, works in the following way.

Its constituents *Frm* and *Ctr* work in parallel and execute *synchronously* actions $a \in \Sigma$. Suppose now that at a given moment frame *Frm* is in a state $s \in S$ and control *Ctr* is in a state $q \in Q$. Then all the clocks of *Ctr* coloured with the colour $\omega_S(s)$ associated with the current state of *Frm* advance with the same speed $\mathrm{vel}(s)$. At the same time, all clocks coloured with colours different from $\omega_S(s)$ are stopped, i.e. they do not change their valuation. Thus we can see that it is the state of the frame that determines which clocks are active and their speed. On the other hand, it is the control automaton that examines the current clock valuation to determine which transitions are executable and determines how valuation is modified by transitions.

Thus $S \times Q$ is the set of states of $\mathcal{A}$ and the set $\Delta_{\mathcal{A}}$ of transitions of $\mathcal{A}$ consists of pairs

$$\delta = (\delta', \delta''), \quad \text{where}$$
$$\delta' = (s, a, s') \in \Delta_{\mathit{Frm}} \ \text{ and } \ \delta'' = (q, a, \varphi, \mathrm{op}, q') \in \Delta_{\mathit{Ctr}} \tag{3}$$

(note that δ' and δ'' should execute the same action $a \in \Sigma$ to form a transition of $\mathcal{A}$).

The behaviour of $\mathcal{A}$ can now be described by means of two mappings.

The first one describes how clocks evolve without any transition: given the current clock valuation $v \in [\![X \to \mathbb{R}]\!]$ and the current state $s \in S$ of *Frm*, by $(v +_s t)$ we shall denote the clock valuation in time $t \in \mathbb{R}_{\geq 0}$ (provided that no

[2] Formally, $\langle -; - \rangle : [\![X \to \mathbb{R}]\!] \times \mathbf{Operations} \longrightarrow [\![X \to \mathbb{R}]\!]$ is a two argument mapping into $[\![X \to \mathbb{R}]\!]$.

transition was executed). From our previous discussion it is clear that for $x \in X$ we have

$$(v +_s t)(x) = \begin{cases} v(x) + \mathrm{vel}(s) \cdot t & \text{if } \omega_X(x) = \omega_S(s) \\ v(x) & \text{otherwise.} \end{cases}$$

The second mapping, $\mathrm{Exec}(-,-) : [\![X \to \mathbb{R}]\!] \times \Delta_{\mathcal{A}} \longrightarrow [\![X \to \mathbb{R}]\!]$ takes as arguments a transition δ and a clock valuation v just before execution of δ and provides the clock valuation $\mathrm{Exec}(v, \delta)$ immediately after execution of δ.

It would be natural to set just $\mathrm{Exec}(v, \delta) = \langle v; \mathrm{op} \rangle$.[3] Unfortunately, with such a definition $\mathcal{A}$ would not satisfy postulate (P1) from Introduction. Roughly speaking, if a clock is allowed to have a positive speed on one occasion and a negative speed on another occasion then it could be used to implement a counter, in particular we could construct an automaton recognizing a language with a non-regular untime projection of the form $\{a^n b c^n \mid n \in \mathbb{N}\}$. To prevent such a possibility we adopt an additional rule for clock reset.

Suppose that v is a clock valuation just before execution of δ and let $v' = \langle v; \mathrm{op} \rangle$. Then the value of clock x just after execution of δ is set to 0 if

$$v'(x) \notin [\mathbf{lower}(x); \mathbf{upper}(x)] \quad \text{and} \tag{4}$$

$$\exists t \in \mathbb{R}_{>0}, \quad (v' +_{s'} t)(x) \in [\mathbf{lower}(x); \mathbf{upper}(x)] \ , \tag{5}$$

i.e. we reset x if after applying op to v the resulting value $v'(x)$ of x is outside of its boundaries (4) but x can attain the interval $[\mathbf{lower}(x); \mathbf{upper}(x)]$ in the future (5) (without any new transition being executed). Let us note that the conjunction of (4) and (5) is equivalent to the following formula:

$$\begin{aligned} \mathrm{Reset}(v, \delta)(x) = (\omega_X(x) = \omega_S(s')) \ \wedge \\ ((\mathrm{vel}(s') < 0 \ \wedge \ \langle v; \mathrm{op} \rangle(x) > \mathbf{upper}(x)) \ \vee \\ (\mathrm{vel}(s') > 0 \ \wedge \ \langle v; \mathrm{op} \rangle(x) < \mathbf{lower}(x))) \ . \end{aligned}$$

(To see this equivalence note that the formula above says that (1) x is active in the state s', which is the frame state attained after execution of δ, and (2) $v'(x) = \langle v; \mathrm{op} \rangle(x)$ is outside of $[\mathbf{lower}(x); \mathbf{upper}(x)]$ but the velocity direction is such that x can enter this interval in the future.)

To summarize our discussion we can define now formally the mapping Exec:

$$\mathrm{Exec}(v, \delta)(x) = \begin{cases} 0 & \text{if} \quad \mathrm{Reset}(v, \delta)(x) \\ \langle v; \mathrm{op} \rangle(x) & \text{otherwise} \ . \end{cases} \tag{6}$$

It is worth noting that any elementary operation over a clock x (either reset or mod) assigns to x a value between $\mathbf{lower}(x)$ and $\mathbf{upper}(x)$, i.e. if op contains any such elementary operation then $\mathrm{Reset}(v, \delta)(x)$ is false and in this case $\mathrm{Exec}(v, \delta)(x) = \langle v; \mathrm{op} \rangle(x)$.

[3] Here and during all the subsequent discussion we assume that δ is of the form (3), i.e. δ is a transition from (s, q) to (s', q') labelled by a, with test φ and executing operation op.

A run of automaton $\mathcal{A} \in \mathrm{CRTA}_{Frm}$ over a timed word $w = (t_0, a_1, t_1, a_2, t_2, \dots)$ is a sequence $r = (s_0, q_0), v_1, (s_1, q_1), v_2, (s_2, q_2), \dots$ of clock valuations v_i and states (s_i, q_i) of $\mathcal{A}$ such that there exists a sequence $\delta_1, \delta_2, \dots$ of transitions of $\mathcal{A}$, where $\forall i, \delta_i = (\delta_i', \delta_i'') \in \Delta_{\mathcal{A}}$, satisfying the following conditions:

- $v_1 = (v_0 +_{s_0} t_0)$, where $v_0(x) = 0$ for all x is the initial clock valuation,
- $v_{i+1} = (\mathrm{Exec}(v_i, \delta_i) +_{s_i} t_i)$ for $i \geq 1$ and,
- $v_i \models \varphi_i$, where φ_i is the test of δ_i'', $i \geq 1$.

Intuitively, v_i in the run r is the clock valuation at the moment when δ_i is executed.

The run r accepts w if either w is finite and the last state of the control *Ctr* in r belongs to Q_{F} or w is infinite and the set of the control states that are visited infinitely often in r belongs to $\mathcal{F}$ (Muller condition, cf.[11]).

A control automaton *Ctr* is *deterministic* if for any pair of transitions $\delta_1 = (q, a, \varphi_1, \mathrm{op}_1, q_1)$ and $\delta_2 = (q, a, \varphi_2, \mathrm{op}_2, q_2)$ and any admissible valuation[4] v, $v \models \varphi_1 \wedge \varphi_2$ implies $\delta_1 = \delta_2$, cf. [2]. A controlled automaton $\mathcal{A} = (Frm, Ctr)$ is deterministic iff both the frame and the control are deterministic.

Remark 1. Without loss of generality we can restrain ourselves to controls *Ctr* with integer constraints (i.e. with only integer constants in tests and in boundaries). The reason is the same as in real-time automata of Alur and Dill: multiplying all constants appearing in *Ctr* by an appropriate integer n we get a new control *Ctr'* with integer constraints such the languages $\mathcal{L}$ and $\mathcal{L}'$ recognized by (Frm, Ctr) and (Frm, Ctr') are related in the following way: timed word $(t_1, a_1, t_2, a_2, t_3, a_3, \dots)$ belongs to $\mathcal{L}$ iff timed word $(nt_1, a_1, nt_2, a_2, nt_3, a_3, \dots)$ belongs to $\mathcal{L}'$. In particular both $\mathcal{L}$ and $\mathcal{L}'$ have the same untime projection. Therefore, from this moment onwards we assume that all constants appearing in controlled automata are integers.

It is time to give some examples of controlled timed automata.

First we shall consider automata with the trivial frame. Such a frame has only one colour and one state (coloured by this unique colour). The velocity mapping for the trivial frame is given by $\mathrm{vel}(s_0) = 1$ (all clocks progress always with the same speed 1). The control automata over the trivial frame have simpler structure since we do not need to specify the colouring mapping ω_X (as all clocks have the same unique colour), and also the boundaries **lower** and **upper** are superfluous since they are never used if the clock velocity does not change the sign.

Example 1. Controlled automata over the trivial frame such that all elementary tests are of the form $(x \# a)$ and only reset operations are admitted coincide with the real-time automata of Alur and Dill [2].

[4] Admissible valuations are defined and discussed later. Let us say now only that, as it turns out, valuations that can appear in runs are admissible.

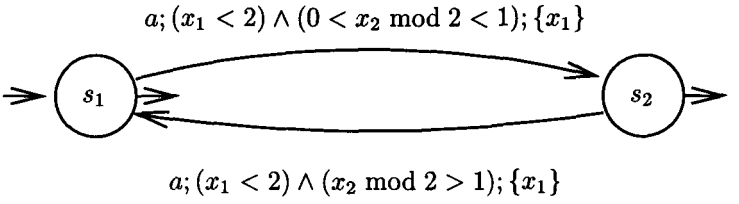

Fig. 1. This deterministic PRTA automaton recognizes the language $\mathcal{L} = \{(a, \tau_1), (a, \tau_2), \ldots, (a, \tau_n) \mid \forall i, (i-1) < \tau_i < i\}$.

Example 2. Let us consider now controlled automata with the trivial frame, however now admitting also the tests of the form $(x \bmod k)\#b$.This is the class of *automata with periodic constraints* (PRTA for short) that was recently examined by Choffrut and Goldwurm [5].

First we can note that deterministic PRTA can recognize languages that are not recognizable by (non-deterministic) RTA. The first adequate example is due to Diekert, Gastin and Petit [6]. However the method of [6] was relatively involved and could not cope with some simple but interesting examples, as the one on Figure 1.

Recently Hermann [9, 10], building partially on some ideas of [6] and [7], provided a simpler and more robust method for proving non-recognizability by RTA. This method allows to prove formally that the language recognized by the deterministic PRTA of Figure 1 cannot be recognized by any non-deterministic RTA. Note however, that if we allow ε-transitions in Alur and Dill's model than we can recognize (non-deterministically) all languages recognized by PRTA.

Now we shall present examples illustrating the expressive power of deterministic controlled timed automata with non-trivial frames. These examples are quite significant since the languages recognized by the presented automata cannot be recognized by non-deterministic RTA even if we allow ε-transitions. This can be contrasted with other previously proposed extensions of real-time automata: Alur and Henzinger [4], Alur, Fix, Henzinger [3] or automata with periodic constraints of Choffrut and Goldwurm [5], which can all be simulated by non-deterministic RTA with ε-transition.

Example 3. Fig. 2(a) shows the frame and the control of an automaton that recognizes the language $\mathcal{L} = \{(a, \tau), (a, 2\tau), \ldots, (a, n\tau) \mid \exists n \in \mathbb{N}, \exists \tau, 0 < \tau < 1\}$.

To illustrate the limits of controlled timed automata we note that the apparently similar language $\mathcal{L}' = \{(a, \tau), (a, 2\tau), \ldots, (a, n\tau) \mid \exists n \in \mathbb{N}, \exists \tau \in \mathbb{R}_{>0}\}$ cannot be recognized by controlled timed automata, the distance between events should be bounded by some constant in order to recognize such words.

Example 4. Let us consider a system with 3 processes P_i, $i = 1, 2, 3$, initially all of them idle. Process P_i is activated by event a_i and deactivated by event b_i. The system terminates by issuing event c. Figure 3 presents a possible activity

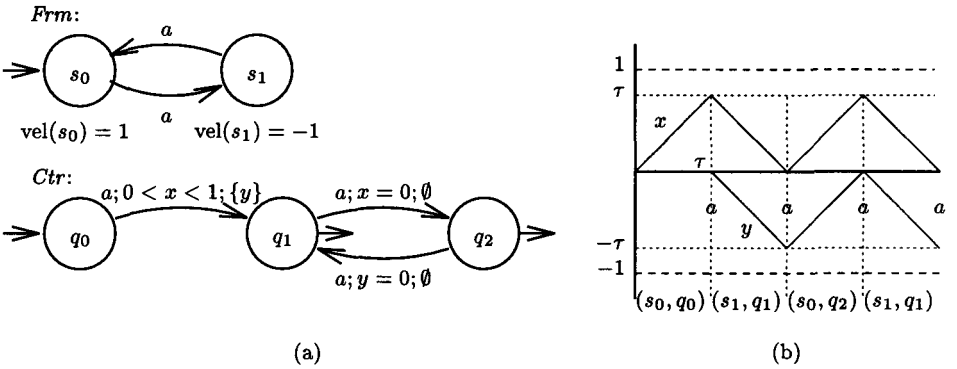

Fig. 2. We assume that there is only one colour colouring both states s_0, s_1 of *Frm* and both clocks x, y, i.e. all clocks are always active. Initially, the control *Ctr* is in the state q_0 and *Frm* in s_0 and both clocks advance with the speed 1. The first event a takes place at any moment $0 < \tau < 1$ and resets the clock y. At the same time the frame passes to the state s_1, thus both clocks change their speed to -1. The second transition happens when the value of x returns to 0, thus the distance between the second and the first transition is τ. At this moment the sign of the speed is reversed again and the whole cycle is repeated (with the roles of x and y inverted). Evolution of clock values is presented on Fig.2(b).

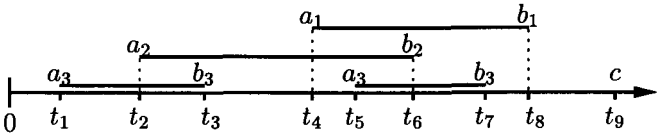

Fig. 3. A possible scenario for the system of Example 4.

scenario (note in particular that processes can be activated and deactivated several times and the activity of different processes can overlap.)

Fig. 4 presents the frame implementing all possible sequences of events in our system. The states of *Frm*, except f, are triples (k_1, k_2, k_3) coding active processes: $k_i = 1$ iff P_i is active.

We impose two timing constraints on our system conditioning the occurrence of event c:

(S1) The total idle time of the system (the time when all processes are idle) should be equal 1, for example the scenario of Fig. 3 should satisfy $(t_1 - 0) + (t_9 - t_8) = 1$,

(S2) Let T_i is the total activity time of P_i, for example for the scenario of Fig.3 $T_1 = t_8 - t_4$, $T_2 = t_6 - t_2$, $T_3 = t_3 - t_1 + t_7 - t_5$. Then we require that $T_1 + T_2 + T_3 = 6$.

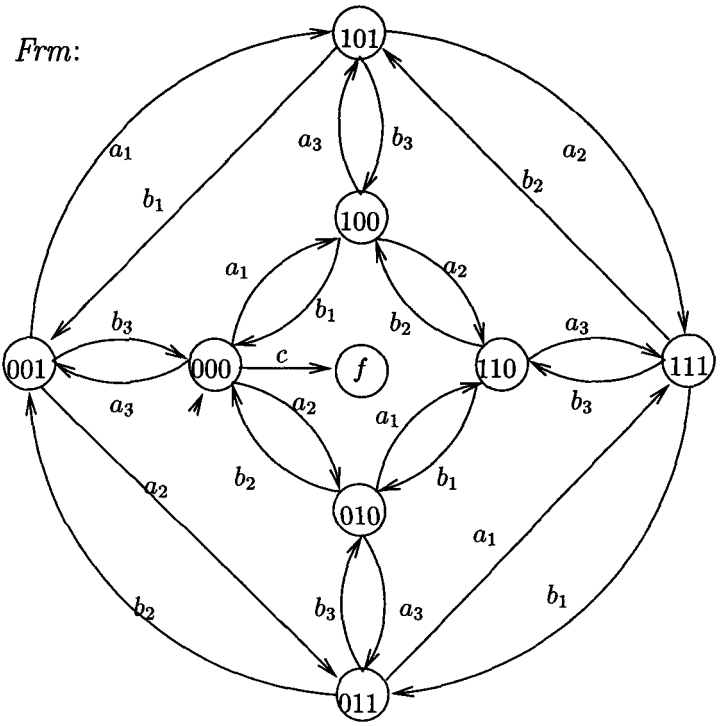

Fig. 4. The frame for the three-process automaton of Example 4.

To implement the timing constraints we use the control automaton presented on Fig. 5. There are two colours $\Omega = \{$"idle","active"$\}$ and two clocks x and y.

Clock x coloured "idle" counts idle time of the system. Since the system is idle if *Frm* is in the state $(0,0,0)$ we assume that the state $(0,0,0)$ of *Frm* is also coloured "idle". Obviously we set also $\mathrm{vel}((0,0,0)) = 1$ to allow x to advance with the speed 1 at $(0,0,0)$.

Clock y is coloured "active" and counts the sum of activity times of processes. Since the are active processes only if the frame is in one of the states $(k_1,k_2,k_3) \neq (0,0,0)$ we colour such states with "active". Since activity of different processes can overlap and condition (S2) concerns the sum of activity times, in order that y correctly counts such sum it suffices to set $\mathrm{vel}(k_1,k_2,k_3) = k_1 + k_2 + k_3$ for all $(k_1,k_2,k_3) \neq (0,0,0)$. In this way we make the velocity of y proportional to the number of active process. For example, for the timed word of Fig.3 in any moment t', $t_4 < t' < t_5$, the frame in the state $(1,1,0)$ (only P_1 and P_2 are active) and the clock y advances with the speed 2.

The following theorem states that controlled timed automata satisfy postulate (P1) from Introduction.

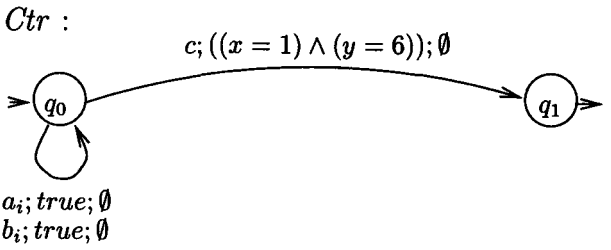

Fig. 5. The control *Ctr* for the automaton of Example 4.

Theorem 1. *For each controlled timed automaton $\mathcal{A}$ the projection* untime($\mathcal{L}(\mathcal{A})$) *on the alphabet Σ is a recognizable word language.*

Proof. Our proof mimics the one of [2]. Let $\mathcal{A}$ be a controlled automaton with a frame (1) and a control (2). A clock valuation v is said to be *admissible* iff there do not exist clocks x, y such that $\omega_X(x) = \omega_X(y)$ and $(v(x) < \mathbf{lower}(x)) \wedge (v(y) > \mathbf{upper}(y))$.

As we shall explain later, due to the special resetting rule implemented in the definition of Exec, all valuations accessible from the initial valuation v_0 are always admissible.

The set of admissible valuations will be denoted by *Val*.

Our first aim is to define an equivalence relation $\approx$ of finite index over *Val*, equivalence classes of $\approx$ are called regions.

First we introduce some necessary notation.

For $v \in Val$, we partition the set of clocks onto three sets: $\mathrm{LOW}_v = \{x \in X \mid v(x) < \mathbf{lower}(x)\}$, $\mathrm{UP}_v = \{x \in X \mid v(x) > \mathbf{upper}(x)\}$, $\mathrm{MID}_v = \{x \in X \mid \mathbf{lower}(x) \leq v(x) \leq \mathbf{upper}(x)\}$.

Note that valuation v is admissible iff $\forall c \in \Omega$ either $\omega_X^{-1}(c) \cap \mathrm{LOW}_v = \emptyset$ or $\omega_X^{-1}(c) \cap \mathrm{UP}_v = \emptyset$.

For each clock $x \in X$, let p_x be the least common multiple of all integers k such that *Ctr* contains either an elementary test of the form $((x \bmod k) \# a)$ or an operation of the form $(x \bmod k)$. (If there is no such k then p_x is undefined.)

With this notation we define $v \approx v'$, for $v, v' \in Val$, iff

(R1) $\mathrm{LOW}_v = \mathrm{LOW}_{v'}$, $\mathrm{MID}_v = \mathrm{MID}_{v'}$ and $\mathrm{UP}_v = \mathrm{UP}_{v'}$,

(R2) for all $x \in X$, if $x \in \mathrm{MID}_v (= \mathrm{MID}_{v'})$ then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ and $\lceil v(x) \rceil = \lceil v'(x) \rceil$,[5]

(R3) $\forall x \in X$, if p_x is well-defined and $x \in \mathrm{LOW}_v \cup \mathrm{UP}_v (= \mathrm{LOW}_{v'} \cup \mathrm{UP}_{v'})$ then $\lfloor v(x) \bmod \mathrm{p}_x \rfloor = \lfloor v'(x) \bmod \mathrm{p}_x \rfloor$ and $\lceil v(x) \bmod \mathrm{p}_x \rceil = \lceil v'(x) \bmod \mathrm{p}_x \rceil$ (this condition is void if p_x is undefined),

(R4) $\forall x, y \in X$, if $\omega_X(x) = \omega_X(y)$ then $\mathrm{fract}(v(x)) \leq \mathrm{fract}(v(y))$ iff $\mathrm{fract}(v'(x)) \leq \mathrm{fract}(v'(y))$.

[5] Note that for $a, b \in \mathbb{R}$, $(\lfloor a \rfloor = \lfloor b \rfloor) \wedge (\lceil a \rceil = \lceil b \rceil)$ is equivalent to $(\lfloor a \rfloor = \lfloor b \rfloor) \wedge (\mathrm{fract}(a) = 0 \Leftrightarrow \mathrm{fract}(b) = 0)$.

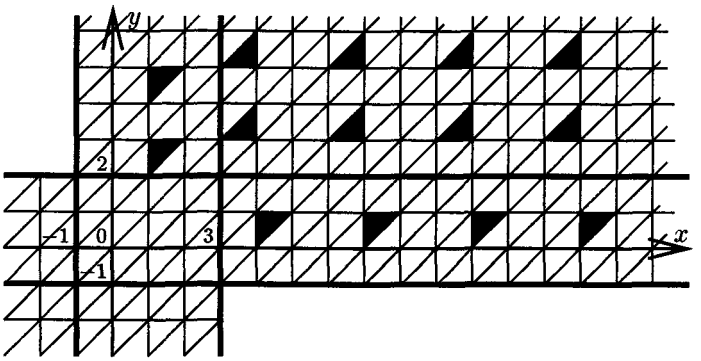

Fig. 6. Regions for two clocks.

We postpone for a while the proof of Theorem 1 in order to illustrate the region definition.

Example 5. Suppose that we have only two clocks x and y, both coloured with the same colour and such that $\mathbf{lower}(x) = -1, \mathbf{upper}(x) = 3, \mathbf{lower}(y) = -1, \mathbf{upper}(y) = 2, \mathrm{p}_x = 3$ and $\mathrm{p}_y = 2$. Figure 6 represents the regions in this situations.

For our discussion it will be convenient to distinguish the following zones:

$$U = \{(x,y) \mid x \geq -1 \text{ and } y \geq -1\}, \quad L = \{(x,y) \mid x \leq 3 \text{ and } y \leq 2\}$$
$$M = L \cap U \ . \quad (7)$$

Inside the rectangle M the regions have the same form as in [2], each small triangle is in fact the union of 7 regions: the triangle interior, its 3 sides (without endpoints), and its 3 vertices.

Outside of M each region is rather an infinite union of either triangle interiors, or triangle sides or triangle vertices.

For example, in the zone $A = \{(x,y) \mid x > 3, y > 2\}$ for each small triangle interior T and any point $p \in K, p = (a,b)$, all triangle interiors $T' \subset A$ containing any of the points of the form $(a + i\,\mathrm{p}_x, b + j\,\mathrm{p}_y)$ constitute one region. The same condition holds for triangle sides and triangle vertices.

For example the interiors of all grey triangles in the zone A on Fig. 6 constitute one region.

For other zones, like $B = \{(x,y) \mid -1 \leq x \leq 3, y > 2\}$ regions are constructed in a similar way. The only difference is that in the case of B only the periodicity along y axis is used to "glue" together small subregions.

Using the same example we can examine also how clock valuations can evolve in time. Inside of the square M (7) the velocity can be either positive or negative. If it positive then (x,y) can enter the zone $U \setminus M$. If $(x,y) \in U \setminus M$ and if a transition changes the velocity to a negative one then the resulting new valuation

will lie inside of M — this is the role of the condition Reset in the definition of Exec to assure that we never visit $U \setminus M$ with negative velocity.

Symmetrically, it is impossible to visit the zone $L \setminus M$ with a positive velocity. Now we can see also that the zone $\mathbb{R}^2 \setminus (U \cup L)$ is never accessible if we start with the initial valuation v_0 corresponding to the point $(0,0)$ on Fig. 6. This explains why we are interested only in admissible valuations.

Now we can resume the proof of Theorem 1. The region automaton $\mathrm{R}(\mathcal{A})$ is a finite automaton over Σ with the states of the form $(s, q, [v]_{\approx})$, where $(s,q) \in S \times Q$, $v \in \mathit{Val}$ and $[v]_{\approx}$ is the equivalence class of v under $\approx$. The initial state equals $(s_0, q_0, [v_0]_{\approx})$. Automaton $\mathrm{R}(\mathcal{A})$ has a transition $(s, q, [v]_{\approx}) \xrightarrow{a} (s', q', [v']_{\approx})$ iff there exists a transition $\delta \in \Delta_{\mathcal{A}}$ of the form (3) such that $v \models \varphi$ and $v' = \mathrm{Exec}(v, \delta)$.

Let $v \approx v_1$. Since test φ uses only constants $c \in [\mathbf{lower}(x); \mathbf{upper}(x)]$, we have $v \models \varphi$ iff $v_1 \models \varphi$. Moreover, direct verification shows that if $v_1' = \mathrm{Exec}(v_1, \delta)$ then $v' \approx v_1'$. Therefore the transition definition given here is unambiguous — it is independent of the choice of the valuation in the class $[v]_{\approx}$.

Automaton $\mathrm{R}(\mathcal{A})$ has also ε transitions simulating region changes due to time progression:

- $(s, q, [v]_{\approx}) \xrightarrow{\varepsilon} (s, q, [v]_{\approx})$ iff $\forall t \in \mathbb{R}_{\geq 0}, v \approx (v +_s t)$,
- if $v_1, v_2 \in \mathit{Val}$ are not equivalent by $\approx$ then $(s, q, [v_1]_{\approx}) \xrightarrow{\varepsilon} (s, q, [v_2]_{\approx})$ iff $\exists t \in \mathbb{R}_{>0}$ such that $v_2 = (v_1 +_s t)$ and $\forall 0 < t' < t$, either $v_1 \approx (v_1 +_s t')$ or $v_2 \approx (v_1 +_s t')$.

Again a direct but long and tedious verification shows that the definition of ε-transitions depends only on the class $[v_1]_{\approx}$ and not on the choice of its member, i.e. if $v_1' \approx v_1$ then there exists v_2' such that $v_2 \approx v_2'$ and $(s, q, [v_1']_{\approx}) \xrightarrow{\varepsilon} (s, q, [v_2']_{\approx})$.

We end the construction of $\mathrm{R}(\mathcal{A})$ by taking $\{(s, q, [v]_{\approx}) \mid q \in Q_{\mathrm{F}}\}$ as the set of final states. The family of infinitely repeated states of $\mathrm{R}(\mathcal{A})$ consists of sets A of states of $\mathrm{R}(\mathcal{A})$ such that $\{q \in Q \mid \exists s \in S, \exists v \in \mathit{Val}, (s, q, [v]_{\approx}) \in A\} \in \mathcal{F}$.

The final and easy step, again similar as in [2], consists in showing that the language recognized by $\mathrm{R}(\mathcal{A})$ equals $\mathrm{untime}(\mathcal{L}(\mathcal{A}))$.

□

Let us note that the construction of the region automata does not depend on the absolute values of clock velocities $|\mathrm{vel}(s)|$, $s \in S$ (but it depends on the sign $\mathrm{sgn}(\mathrm{vel}(s))$.
Therefore we deduce that

Remark 2. If $\mathit{Frm} = (\Sigma, \Omega, S, s_0, \Delta_{\mathit{Frm}}, \omega, \mathrm{vel})$ is a frame and $\mathrm{vel}' : S \to \mathbb{Q}$ is another velocity mapping such that $\forall s \in S, \mathrm{vel}(s) \cdot \mathrm{vel}'(s) > 0$, then for each Ctr and for the frame $\mathit{Frm}' = (\Sigma, \Omega, S, s_0, \Delta_{\mathit{Frm}}, \omega, \mathrm{vel}')$ we have $\mathrm{untime}(\mathcal{L}((\mathit{Frm}', \mathit{Ctr}))) = \mathrm{untime}(\mathcal{L}((\mathit{Frm}, \mathit{Ctr})))$.

The proof of the following theorem is standard (and omitted).

Theorem 2. *Let Frm be a fixed deterministic frame. Then*

(1) the class of languages recognized by deterministic controlled automata with frame Frm is closed under boolean operations,
(2) for any two (not necessarily deterministic) controls Ctr_1, Ctr_2 there exist controls Ctr', Ctr" such that $\mathcal{L}(Frm, Ctr') = \mathcal{L}(Frm, Ctr_1) \cap \mathcal{L}(Frm, Ctr_2)$ and $\mathcal{L}(Frm, Ctr'') = \mathcal{L}(Frm, Ctr_1) \cup \mathcal{L}(Frm, Ctr_2)$ (for the union the condition that Frm is deterministic is superfluous).

Remark 3. Theorems 1 and 2 show only that for each deterministic frame *Frm* the class CRTA_{Frm} satisfies postulates (P1)–(P2). Can the union of these classes satisfy (P1)-(P2) as well? Unfortunately, this is impossible. It turns out that there exist deterministic controlled automata $\mathcal{A}_1 = (Frm_1, Ctr_1)$ and $\mathcal{A}_2 = (Frm_2, Ctr_2)$, over the same signature but with different frames $Frm_1 \neq Frm_2$, such that $\mathrm{untime}(\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2))$ is not a recognizable language of Σ^*.

4 Final remarks.

At first it may seem that various constraints built into the definition of controlled automata are quite arbitrary. Let us list some of them:
(1) Clock colour classes are always disjoint (i.e. no clock can be coloured with more than one colour), (2) at a given moment clocks in the same colour class have the same velocity, (3) at a given moment only clocks of one colour class are allowed to progress. As it turns out relaxing any one of these conditions yields automata with non-regular untime projection.

Another possible extension consists in allowing elementary tests of the form $x\#y$, $x\#(y \bmod k)$ or $(x \bmod k_1)\#(y \bmod k_2)$, $x, y \in X$. If limited to pairs of clocks coloured with the same colour then such extension does not increase the expressibility power of controlled automata. On the other hand, allowing any of these tests for pairs of clocks with different colours would yield again automata with non-regular untime projection.

References

1. Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in real-time systems. *Information and Computation*, 104:2–34, 1993.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. Rajeev Alur, Limor Fix, and Thomas A. Henzinger. A determinizable class of timed automata. In *Proceedings of CAV'94*, volume 818 of *Lecture Notes in Comp. Sci.*, pages 1–13. Springer Verlag, 1994.
4. Rajeev Alur and Thomas A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *Proceed. of 33th IEEE Symp. of Fundamentals of Computer Science*, pages 177–186, 1992.
5. Christian Choffrut and Massimiliano Goldwurm. Timed automata with periodic clock constraints. Technical report, University of Milano, 1998.
6. V. Diekert, P. Gastin, and A. Petit. Removing ε-transitions in timed automata. In *Proceedings of STACS'97*, volume 1200 of *Lecture Notes in Comp. Sci.*, pages 583–594. Springer Verlag, 1997.

7. V. Gupta, T.A. Henzinger, and R. Jagadeesan. Robust timed automata. In *Proceedings of HART'97*, volume 1201 of *Lecture Notes in Comp. Sci.*, pages 331–345. Springer Verlag, 1997.
8. T.A. Henzinger. The theory of hybrid automata. In *Proceedings of LICS'96*, pages 278–292, 1996.
9. Philippe Hermann. Automates temporisés et reconnaissabilité. Mémoire de DEA, LIAFA, Université Paris VII, 1997. extended version of [10].
10. Philippe Hermann. Timed automata and recognizability. *Information Processing Letters*, 1998. to appear.
11. W. Thomas. Automata on infinite objects. In J. van Leeuven, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science, 1990.
12. Th. Wilke. Specifying timed step sequences in powerful decidable logics and timed automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lecture Notes in Comp. Sci.* Springer Verlag, 1994.