Timed automata and additive clock constraints

Béatrice Bérard and Catherine Dufourd

LSV, CNRS UMR 8643, ENS de Cachan, 61 av. du Prés. Wilson,
94235 Cachan cedex, France
E-mails: {berard,dufourd}@lsv.ens-cachan.fr

Keywords: Real-time systems, timed automata, decidability of emptiness

1 Introduction

The model of timed automata, introduced in [1], is obtained from classical finite automata by adding a finite set of real valued variables called clocks. Clock values increase continuously at the same rate as time in the control locations, and they can be tested and reset by transitions. A test consists in comparing clock values, or the difference between two such values, with constants. For these models, the test for emptiness is decidable (and PSpace-complete [2]), which explains their successful use for the verification of timed systems.

Extensions have later been proposed in several directions, with the aim to increase the expressive power, while preserving the decidability result. For instance, decidability of emptiness still holds for some classes of timed transition systems (like Petri nets or context-free grammars), where the hypothesis of finite control is removed ([4]). Replacing reset by more general update operations can also preserve decidability ([6]). Another extension, yielding so-called linear hybrid systems [8], is obtained by adding variables with different (rational) slopes and allowing linear inequalities over their values. When these variables are controlled by a finite timed automaton as in [7], emptiness remains decidable. However, in the general setting, emptiness is undecidable ([9]).

In this note, we consider the subclass of linear hybrid automata, which consists of timed automata extended with additive clock constraints. In [2], this model is proved to be strictly more expressive than the basic one and the test for emptiness becomes undecidable. We extend this result to timed automata with only 4 clocks and a restricted form of additive constraints.

2 Timed automata with additive constraints

The set of clock constraints over a set X of clocks is written $\mathcal{C}(X)$ and consists of boolean combinations of atomic propositions of the form: $x \# c$, $x + y \# c$, where x, y are clocks, $c \in \mathbb{N}$ is a constant and $\#$ is a binary operator in $\{<, =, >\}$.

A clock valuation is a mapping $v : X \mapsto \mathbb{R}_+$. The set of all clock valuations is $\mathbb{R}_+^X$ and we write $v \models \varphi$ when valuation v satisfies the clock constraint φ. For an element t of $\mathbb{R}_+$ and a subset α of X, the valuations $v + t$ and $v[\alpha \leftarrow 0]$ are defined respectively by $(v + t)(x) = v(x) + t$, for each clock x in X and $(v[\alpha \leftarrow 0])(x) = 0$ if $x \in \alpha$, $v(x)$ otherwise.

A timed automaton is a tuple $\mathcal{A} = (Q, X, \Sigma, \Delta, q_0, F)$, where Q is a finite set of locations, X is a finite set of clocks, Σ is a finite alphabet, $\Delta \subseteq Q \times [\mathcal{C}(X) \times \Sigma \times \mathcal{P}(X)] \times Q$ is the set of transitions, q_0 is the initial state and $F \subseteq Q$ is the set of final states. A transition $(q, \varphi, a, \alpha, q')$ in Δ, also written $q \xrightarrow{\varphi, a, \alpha} q'$, contains a clock constraint φ, a label a and a subset $\alpha \subseteq X$ of clocks to be reset.

The semantics of a timed automaton $\mathcal{A}$ are described by the transition system $\mathcal{T}_{\mathcal{A}} = (S, \mapsto)$. The set S of states is $Q \times \mathbb{R}_+^X$, with initial state the pair (q_0, v_0) (initial value 0 for all clocks). An accepting state of $\mathcal{T}_{\mathcal{A}}$ is a pair (q, v) with q in F. The relation $\mapsto$ is defined on S by the following condition:

$(q, v) \xrightarrow{a,t} (q', v')$ if there exists a transition $q \xrightarrow{\varphi, a, \alpha} q'$ in Δ, such that $v + t \models \varphi$, and $v' = v[\alpha \leftarrow 0]$.

A timed word is a sequence $w = (a_1, t_1) \ldots (a_n, t_n) \in (\Sigma \times \mathbb{R}_+)^*$, where the t_i's form a non decreasing sequence. The timed language accepted by $\mathcal{A}$ is the set of timed words w which label some path in $\mathcal{T}_{\mathcal{A}}$ from the initial to an accepting state.

In [2], the undecidability result is proved for timed automata with atomic clock constraints of the form $x + y \ \# \ x' + y'$, while we consider here a restricted subset of additive constraints, of the form $x + y \# c$. In this framework, there is a particular case for timed automata with only two clocks: the corresponding class is strictly more expressive than the usual one, but emptiness remains decidable.

Proposition 1. *Let TL_2^+ be the class of timed languages accepted by timed automata with two clocks and atomic constraints in $\{x \# c,\ x - y \# c,\ x + y \# c\}$.*

1. *Emptiness is decidable in TL_2^+.*
2. *Let $L = \{(a^n, t_1 \ldots t_n) \mid n \geq 1$ and $t_i = 1 - 1/2^i,\ 1 \leq i \leq n\}$. This language belongs to TL_2^+ but there is no timed automaton with constraints built from $\{x \# c,\ x - y \# c\}$ accepting L.*

Proof. Point 1 results from an easy extension of the classical region construction. We fix a maximal value M for the constants appearing in the clock constraints and we have to build a finite partition of $\mathbb{R}_+^X$, which is (i) consistent with the set of constraints (two equivalent valuations satisfy the same atomic constraints with bound M), (ii) consistent with time progression (from two equivalent valuations, the same regions are reached when time increases) and (iii) consistent with reset operations (reset from two equivalent valuations yield the same region).
Figure 1 shows the partition of $\mathbb{R}_+^2$ obtained for $M = 3$.

The language L defined in Point 2 is accepted by the automaton in figure 2. The formal proof that no classical timed automaton can accept L relies on a method introduced in [5]. Let $\mathcal{A}$ be a timed automaton and let K be the common denominator of the (finitely many rational) constants used in $\mathcal{A}$. For a path $\pi = q_0 \xrightarrow{\varphi_1, a_1, \alpha_1} q_1 \xrightarrow{\varphi_2, a_2, \alpha_2} q_2 \cdots$ in $\mathcal{A}$, $TS(\pi)$ is the set of time sequences $t_1 \ldots t_n$ possible through this path (i.e. for which $(a_1, t_1) \ldots (a_n, t_n)$ is accepted by $\mathcal{A}$ through π) and $TS_i(\pi) = \{r \in \mathbb{R}_+ \mid r = t_i$ for some $t_1 \ldots t_n \in TS(\pi)\}$. The occurrence a_i of π is called a precise action in π if $TS_i(\pi)$ reduces to a singleton. Applying first part of theorem 9.1 in [5] shows that in this case, the single value t_i in $TS_i(\pi)$ belongs to $K\mathbb{N}$.

Now suppose that $\mathcal{A}$ accepts L. For each path π in $\mathcal{A}$, $TS(\pi)$ contains at most one time sequence, so that all occurrences of a are precise. Hence we should have $1 - 1/2^i \in K\mathbb{N}$ for any $i \geq 1$, which is of course impossible.

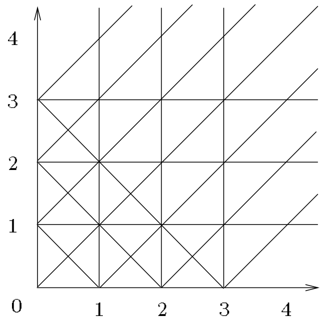

Fig. 1. Regions for constraints $\{x \# c,\ x - y \# c,\ x + y \# c\}$

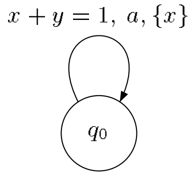

Fig. 2. A timed automaton using constraint $x + y = 1$ for L

We show in the next section that emptiness is undecidable for a set of 4 clocks, with atomic constraints of the form $x \# c$ or $x + y = 1$. This extends the general result and gives an upper bound on the number of clocks for which emptiness remains decidable.

3 Undecidability proof

Theorem 2. *For the class TL_4^+ of timed languages accepted by timed automata with four clocks, extended with additive clock constraints of the form $x + y = 1$, the test for emptiness is undecidable.*

Proof. The proof lies in the simulation of a deterministic two counters machine. Recall that such a machine $\mathcal{M}$ consists of a finite sequence of labeled instructions, which handle two counters c and d, and end at a special instruction with label *Halt*. The other instructions have one of the two forms below, where $x \in \{c, d\}$ represents one of the two counters:

1. $\ell : x := x + 1; goto\ \ell'$;
2. $\ell : if\ x = 0\ goto\ \ell'\ else\ x := x - 1;\ goto\ \ell''$;

Without loss of generality, we may assume that the counters have initial value zero. The behaviour of the machine is described by a (possibly infinite) sequence of configurations: $< \ell_0, 0, 0 >< \ell_1, n_1, m_1 > \cdots < \ell_i, n_i, m_i > \cdots$, where n_i and m_i are the respective counter values and ℓ_i is the label, after the i^{th} instruction. The problem of termination for such a machine ("is the *Halt* label reached ?") is known to be undecidable ([10]).
Our aim is to obtain a reduction of this problem to the problem of emptiness for a language in TL_4^+. We build a timed automaton $\mathcal{A}_{\mathcal{M}}$ with 4 clocks which simulates the two counters machine $\mathcal{M}$ and reaches a final location if and only if $\mathcal{M}$ stops.

Alphabet. The automaton $\mathcal{A}_{\mathcal{M}}$ has alphabet $\Sigma = \{c, d, *\}$.
It has no ε-transition, but uses instead the letter $*$, which is the label of an internal but observable action. A value n for counter $x \in \{c, d\}$ is encoded in a (partial) timed word $w = (x^n, t_1 \ldots t_n)$, where the time sequence is of the form $p - 1/2, p - 1/2^2, \ldots, p - 1/2^n$, inside an interval $[p-1, p]$ of length 1. Such a word can be seen as a copy (with a time shift) of a word in the language L from proposition 1. The values of counters c and d alternate when time progresses: for each $i \geq 0$, n_i letters c can be read within time interval $[2i, 2i+1]$ and m_i letters d within $[2i+1, 2(i+1)]$.
Note that a timed action of the form $(*, 1)$ occurs at the beginning (see figure 4), and other timed actions $(*, t)$ for some t, appear before and/or after each partial timed word coding a counter value. They are omitted in figure 3, where some configuration $< \ell, 3, 4 >$ is represented by the word $(c, 20+1/2)(c, 20+3/4)(c, 20+7/8)(d, 21+1/2)(d, 21+3/4)(d, 21+7/8)(d, 21+15/16)$.

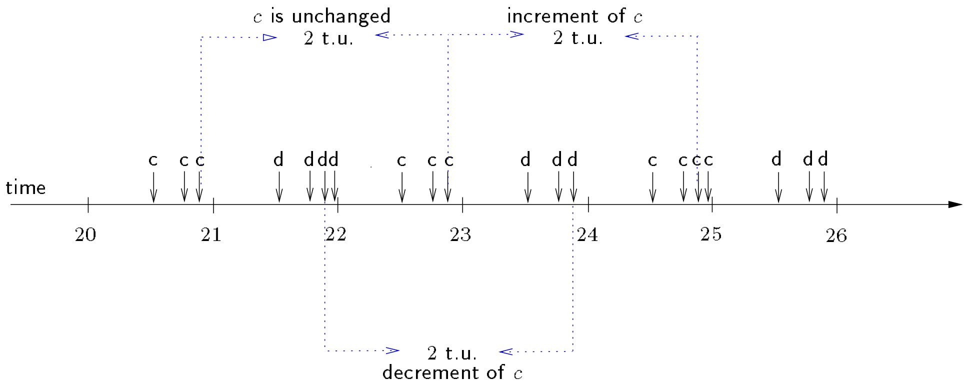

Fig. 3. Simulation of sub-sequence $\cdots < 3, 4 >< 3, 3 >< 4, 3 > \cdots$

Clocks. The set of clocks is $X = \{u, x_0, x_1, x_2\}$. The clock u is used to obtain the time intervals of length 1, so that it is reset each time it reaches the value 1. Each of the three clocks x_i plays a specific role within one interval. However, in order to reduce the total number of clocks, the x_i's will exchange their roles at the end of each interval, as explained later.

- One of the x_i's computes the dates associated with the letter (c or d) currently treated. It is reset at the beginning of the interval and, as in the automaton for L in figure 2 above, the transition labeled by the letter is executed if the constraint $u + x_i = 1$ holds, after which x_i is reset. As for L but on a given interval of length 1, the automaton accepts an arbitrary number of letters, each one occurs after one half of the time that remains to reach the beginning of the next interval. Figure 3 shows the dates for some sub-sequence $< 3, 4 >< 3, 3 >< 4, 3 >$, where the labels have been omitted.
- Another one of the x_i's is used to stop the current counting of a letter, by comparison with the constant 2. We call this clock the *reference* of the letter.
- The last clock is not useful within the current interval, but plays the same role of reference for the other letter in the next interval.

Basic modules. The set of locations of $\mathcal{A}_\mathcal{M}$ is:
$Q = \{(\ell, x_i) \mid i \in \{0, 1, 2\}\} \cup \{loop(\ell, x_i) \mid i \in \{0, 1, 2\}\} \cup \{q_0\}$, where the initial location is q_0 and the set of final locations is $F = \{(Halt, x_i) \mid i \in \{0, 1, 2\}\}$. The transitions of $\mathcal{A}_\mathcal{M}$ are described by parts of this automaton, called *modules.*

Initialization of the machine is simulated by the module drawn in figure 4. Counters c and d have both initial value 0, but the first clock reference for c is x_0 while the first clock reference for d is x_1. Location (ℓ_0, x_0) is reached at time 1 and simulation of the initial instruction ℓ_0 starts at time 2.

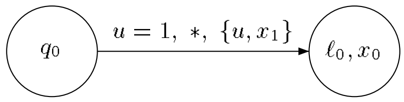

Fig. 4. Initialization module

Three kinds of actions must then be performed to count values of c and d: increment or decrement a counter, or keep its value unchanged, and each action is implemented in $\mathcal{A}_\mathcal{M}$ by three copies of a specific module, one for each x_i ($i \in \{0, 1, 2\}$). A module associated with instruction ℓ and clock x_i has for entry location the pair (ℓ, x_i). Exit locations are defined as follows, where $i + 1$ implicitly uses addition modulo 3:

- an *unchange counter module* (see figure 5) has one exit location (ℓ, x_{i+1}),
- an *increment module* (see figure 6) has one exit location (ℓ', x_{i+1}),
- a *zero testing (or decrement) module* (see figure 7) has two exit locations (ℓ', x_{i+1}) and (ℓ'', x_{i+1}).

We now explain in details how these modules work. Note that indices ℓ and x_0 associated with the *loop* location are omitted in the figures.

Figure 5 is an unchange counter module for c, with entry location (ℓ, x_0). This indicates that x_0 is the reference clock used to ensure that the counting of c is over. In fact, x_0 has been reset with the last letter of the counting of c, two intervals ago. The current interval starts when $u = 1$, then u and x_2 are reset. In the general case $(c \geq 1)$, the loop produces the successive delays $1/2, 1/4, 1/8 \cdots$ as in L, as long as $x_0 < 2$. When $x_0 = 2$, the exit location is reached with the last c. In the special case where $c = 0$, the exit location is directly reached. In both cases, clock x_2 is reset by the last transition and will be the next reference for c, two intervals later. Clock x_1, which is not currently used, will be the reference for letter d, in the next interval.

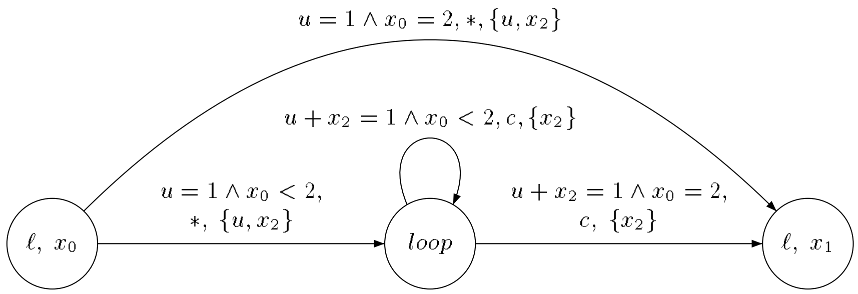

Fig. 5. c is unchanged : reference of c is x_0, next reference of c will be x_2

Figure 6 is an increment module for counter c, with entry location (ℓ, x_0). This module is very similar to the unchange module except there is now one more letter c to count than two intervals ago. The loop is performed as long as $x_0 \leq 2$ and one more c occurs before reaching the exit point. At the same time x_2 is reset and becomes the next reference for c.

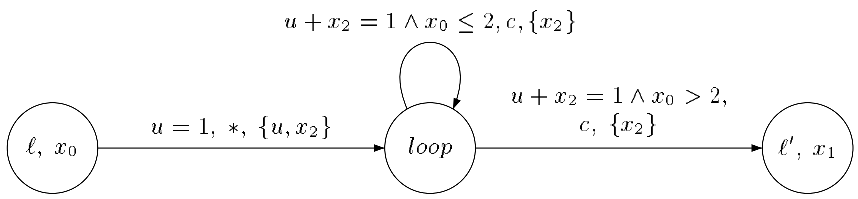

Fig. 6. Increment of c : reference of c is x_0, next reference of c will be x_2

Figure 7 is a zero testing or decrement module for counter c, with entry location (ℓ, x_0). There are two cases. If the value of counter c is zero then we have simultaneously $u = 1$ and $x_0 = 2$, leading to the exit location (ℓ', x_1), after reset of x_2. If counter c is greater than zero then a decrement is simulated. We have $x_0 < 2$ and the transition going to the loop can be fired with reset of u and x_2. Count of c is performed in the loop as long as $x_0 < 2$ and is over when $x_0 = 2$, leading to the exit location (ℓ'', x_1). However, in this case, x_2 is not reset because one c less than two intervals ago is needed. Clock x_2 will be the next reference for c.

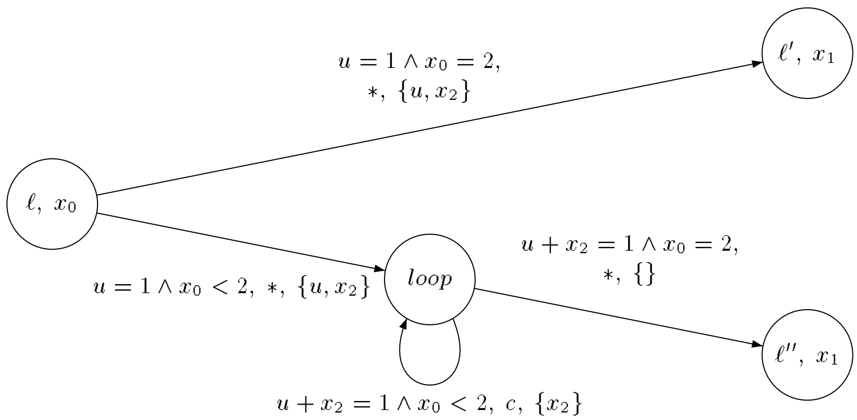

Fig. 7. c is tested for zero : reference of c is x_0, next reference of c will be x_2

Note that for increment and unchange modules, the same clock x_0 could be used for the next reference of c. However, this is not possible with a decrement module, in which the roles of the x_i's must be exchanged: when x_0 reaches value 2 then it is too late to use it for next reference of c. There is no other choice than using x_2, which forces to exchange the roles of x_0 and x_2.

Simulation of the 2-counter machine. To simulate $\mathcal{M}$, the automaton $\mathcal{A}_\mathcal{M}$ starts in location q_0 and for each instruction, performs one count for c followed by one for d. Each instruction is thus implemented with two modules linked together. For instance, an instruction incrementing d is implemented in $\mathcal{A}_\mathcal{M}$ by an unchange module for c linked with an increment module for d. Testing c for zero is implemented by a zero testing module for c linked with an unchange module for d. This "link" is simply obtained in our construction by the fact that an exit location of some module is also the entry location of another one. For instance, an instruction ℓ incrementing d has three possible implementations for $i \in \{0, 1, 2\}$, represented in figure 8.

The automaton $\mathcal{A}_\mathcal{M}$ is simply built according to the sequence of instructions of the 2-counter machine, with each possible implementation of an instruction appearing at most once. Assume that the number of instructions of $\mathcal{M}$ is n, then the number of modules of the associated timed automaton is at most $6n+1$

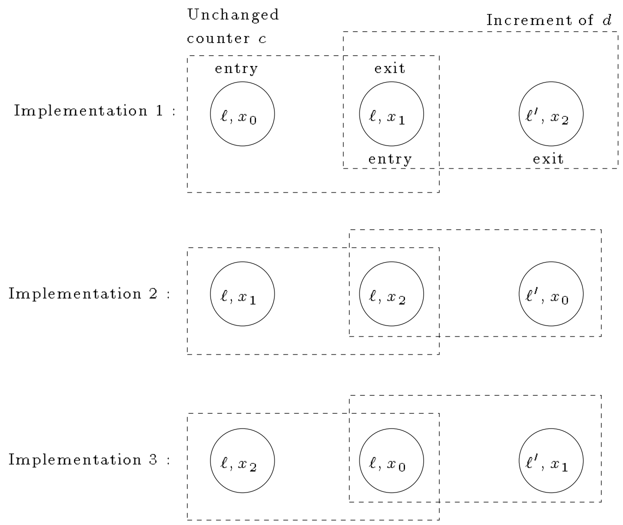

Fig. 8. The three implementations of instruction ℓ: increment of d

(including the initialization module), with the same maximal number of locations (entry and loop for each pair (ℓ, x_i) and initial location).

With the set of final locations $F = \{(Halt, x_i) \mid i \in \{0,1,2\}\}$, the language accepted by $\mathcal{A}_\mathcal{M}$ is empty if and only if $\mathcal{M}$ does not terminate, which concludes the proof.

4 Discussion and Conclusion

We proved that emptiness is undecidable for the class TL_4^+ of languages accepted by four clocks timed automata using constraints of the form $\{x\#c,\ x+y=1\}$. Remark that our construction if quite "minimal". Firstly, we compare $x+y$ to constant 1 only, and not to clocks $(x+y=z)$. Secondly, we do not use *diagonal comparisons* of the kind $x-y\#c$. Such diagonal comparisons often lead to undecidability results for some extended classes of timed automata, while the corresponding classes remain decidable without diagonal constraints ([6]).

We also proved that for the class TL_2^+ corresponding to timed automata with two clocks using constraints $\{x\#c,\ x-y\#c,\ x+y\#c\}$, emptiness is decidable even though this class is strictly more expressive than the one without additive constraints.

The problem remains open for *three* clocks. As for any subclass of linear hybrid automata, decidability of emptiness in TL_3^+ would immediately result from finding a partition of $\mathbb{R}_+^3$ with the consistency properties (i), (ii) and (iii) described in the proof of proposition 1. On the other hand, it also seems difficult to simulate more than one counter with 3 clocks.

An application of our undecidability result concerns another kind of clocks, which we call *countdown* clocks: they are set at a given constant when updated (instead of being reset), and then decrease while time elapses, until reaching 0. Some variants have been studied for instance in [3] and [7] and they can be useful for modelisation purposes (for instance the countdown of a target). From theorem 2, we immediately obtain that emptiness is undecidable for timed automata extended with countdown clocks and using constraints in $\{x-y\#c,\ x\#c\}$, where x, y are either classical or countdown clocks. This relies on the the fact that a constraint $x + y = 1$ is easily implemented with a constraint $x - y_1 = 0$, where y_1 is a countdown clock set to 1 when updated. However, note that for timed automata with countdown clocks and constraints of the form $x\#c$ only, emptiness is still decidable. This is because a countdown clock y_c can be implemented by a classical clock y, replacing any $y_c\#c'$ by $y\#c' + c$. We thus give evidence of a new class for which diagonal constraints lie at the frontier of what is decidable or not.

References

1. R. Alur, D.L. Dill. Automata for modeling real-time systems. In *Proceedings of ICALP'90*, number 443 in Lecture Notes in Computer Science, p:322-335. Springer-Verlag, 1990.
2. R. Alur, D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183-235, 1994.
3. R. Alur, L. Fix, T.A. Henzinger. A determinizable class of timed automata. In *Proceedings of CAV'94*, number 818 in Lecture Notes in Computer Science, p:1-13. Springer-Verlag, 1994.
4. B. Bérard. Untiming timed languages. *Information Processing Letters*, 55:129-135, 1995.
5. B. Bérard, V. Diekert, P. Gastin, A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informatica* 36:145-182, 1998.
6. P. Bouyer, C. Dufourd, E. Fleury, A. Petit. Are timed automata updatable ? In *Proceedings of CAV'00*, to appear.
7. F. Demichelis, W. Zielonka. Controlled Timed Automata. In *Proceedings of CONCUR'98*, number 1466 in Lecture Notes in Computer Science, p:455-469. Springer-Verlag, 1998.
8. T.A. Henzinger. The theory of hybrid automata. In *Proceedings of LICS'96*, p:278-292, 1996.
9. T.A. Henzinger, P. Kopke, A. Puri, P. Varaiya. What's decidable about hybrid automata. In *Proceedings of STOC'95*, p:373-382, 1995.
10. M. Minski. Computation : Finite and Infinite Machines. Prentice Hall, 1967.