Are Timed Automata Updatable ?

Patricia Bouyer, Catherine Dufourd,
Emmanuel Fleury, and Antoine Petit *

LSV, CNRS UMR 8643, ENS de Cachan,
61 Av. du Président Wilson,
94235 Cachan Cedex, France
{bouyer, dufourd, fleury, petit}@lsv.ens-cachan.fr

Abstract. In classical timed automata, as defined by Alur and Dill [AD90,AD94] and since widely studied, the only operation allowed to modify the clocks is the reset operation. For instance, a clock can neither be set to a non-null constant value, nor be set to the value of another clock nor, in a non-deterministic way, to some value lower or higher than a given constant. In this paper we study in details such updates.
We characterize in a thin way the frontier between decidability and undecidability. Our main contributions are the following :

- We exhibit many classes of updates for which emptiness is undecidable. These classes depend on the clock constraints that are used – diagonal-free or not – whereas it is well known that these two kinds of constraints are equivalent for classical timed automata.
- We propose a generalization of the region automaton proposed by Alur and Dill, allowing to handle larger classes of updates. The complexity of the decision procedure remains Pspace-complete.

1 Introduction

Since their introduction by Alur and Dill [AD90,AD94], timed automata are one of the most studied models for real-time systems. Numerous works have been devoted to the "theoretical" comprehension of timed automata and their extensions (among a lot of them, see [ACD+92], [AHV93], [AFH94], [ACH94], [Wil94], [HKWT95], [BD00], [BDGP98]) and several model-checkers are now available (HyTech[1] [HHWT95,HHWT97], Kronos[2] [Yov97], Uppaal[3] [LPY97]). These works have allowed to treat a lot of case studies (see the web pages of the tools) and it is precisely one of them – the ABR protocol [BF99,BFKM99] – which has motivated the present work. Indeed, the most simple and natural modelization of the ABR protocol uses updates which are not allowed in classical timed automata, where the only authorized operations on clocks are resets. Therefore we

* This work has been partly supported by the french project RNRT "Calife"

1 http://www-cad.eecs.berkeley.edu/~tah/HyTech/

2 http://www-verimag.imag.fr/TEMPORISE/kronos/

3 http://www.docs.uu.se/docs/rtmv/uppaal

have considered updates constructed from simple updates of one of the following forms:

$x :\sim c \ | \ x :\sim y + c$, where x, y are clocks, $c \in \mathbb{Q}_+$, and $\sim \in \{<, \leq, =, \neq, \geq, >\}$

More precisely, we have studied the (un)decidability of the emptiness problem for the extended timed automata constructed with such updates. We call these new automata *updatable timed automata*. We have characterized in a thin way the frontier between classes of updatable timed automata for which emptiness is decidable or not. Our main results are the following :

- We exhibit many classes of updates for which emptiness is undecidable. A surprising result is that these classes depend on the clock constraints that are used – diagonal-free (*i.e.* where the only allowed comparisons are between a clock and a constant) or not (where the difference of two clocks can also be compared with a constant). This point makes an important difference with "classical" timed automata for which it is well known that these two kinds of constraints are equivalent.
- We propose a generalization of the region automaton proposed by Alur and Dill, which allows to handle large classes of updates. We thus construct an (untimed) automaton which recognizes the untimed language of the considered timed automaton. The complexity of this decision procedure remains Pspace-complete.

  Note that these decidable classes are not more powerful than classical timed automata in the sense that for any updatable timed automaton of such a class, a classical timed automaton (with $\varepsilon-$transitions) recognizing the same language – and even most often bisimilar – can be effectively constructed. But in most cases, an exponential blow-up seems unavoidable and thus a transformation into a classical timed automaton can not be used to obtain an efficient decision procedure. These constructions of equivalent automata are available in [BDFP00b].

The paper is organized as follows. In section 2, we present basic definitions of clock constraints, updates and updatable timed automata, generalizing classical definitions of Alur and Dill. The emptiness problem is briefly introduced in section 3. Section 4 is devoted to our undecidability results. In section 5, we propose a generalization of the region automaton defined by Alur and Dill. We then use this procedure in sections 6 (resp. 7) to exhibit large classes of updatable timed automata using diagonal-free clock constraints (resp. arbitrary clock constraints) for which emptiness is decidable. A short conclusion summarizes our results.

For lack of space, this paper does not contain proofs which can be found in [BDFP00a].

2 About Updatable Timed Automata

In this section, we briefly recall some basic definitions before introducing an extension of the timed automata, initially defined by Alur and Dill [AD90,AD94].

2.1 Timed words and clocks

If Z is any set, let Z^* (resp. Z^ω) be the set of *finite* (resp. *infinite*) sequences of elements in Z. And let $Z^\infty = Z^* \cup Z^\omega$.
In this paper, we consider $\mathbb{T}$ as time domain, $\mathbb{Q}_+$ as the set of non-negative rational and Σ as a finite set of *actions*. A *time sequence* over $\mathbb{T}$ is a finite or infinite non decreasing sequence $\tau = (t_i)_{i\geq 1} \in \mathbb{T}^\infty$. A *timed word* $\omega = (a_i, t_i)_{i\geq 1}$ is an element of $(\Sigma \times \mathbb{T})^\infty$, also written as a pair $\omega = (\sigma, \tau)$, where $\sigma = (a_i)_{i\geq 1}$ is a word in Σ^∞ and $\tau = (t_i)_{i\geq 1}$ a time sequence in $\mathbb{T}^\infty$ of same length.
We consider an at most countable set $\mathbb{X}$ of variables, called *clocks*. A clock valuation over $\mathbb{X}$ is a mapping $v : \mathbb{X} \to \mathbb{T}$ that assigns to each clock a time value. The set of all clock valuations over $\mathbb{X}$ is denoted $\mathbb{T}^{\mathbb{X}}$. Let $t \in \mathbb{T}$, the valuation $v + t$ is defined by $(v + t)(x) = v(x) + t$, $\forall x \in \mathbb{X}$.

2.2 Clock constraints

Given a subset of clocks $X \subseteq \mathbb{X}$, we introduce two sets of clock constraints over X. The most general one, denoted by $\mathcal{C}(X)$, is defined by the following grammar:

$$\varphi ::= x \sim c \,|\, x - y \sim c \,|\, \varphi \wedge \varphi \,|\, \neg\, \varphi \,|\, true$$
$$\text{where } x,\, y \in X,\, c \in \mathbb{Q}_+,\ \sim\, \in \{<, \leq, =, \neq, \geq, >\}$$

We will also use the proper subset of *diagonal-free* constraints, denoted by $\mathcal{C}_{df}(X)$, where the comparison between two clocks is not allowed. This set is defined by the grammar:

$$\varphi ::= x \sim c \,|\, \varphi \wedge \varphi \,|\, \neg\, \varphi \,|\, true,$$
$$\text{where } x \in X,\, c \in \mathbb{Q}_+ \text{ and } \sim\, \in \{<, \leq, =, \neq, \geq, >\}$$

We write $v \models \varphi$ when the clock valuation v satisfies the clock constraint φ.

2.3 Updates

An *update* is a function from $\mathbb{T}^{\mathbb{X}}$ to $\mathcal{P}(\mathbb{T}^{\mathbb{X}})$ which assigns to each valuation a set of valuations. In this work, we restrict ourselves to local updates which are defined in the following way.

A *simple update* over a clock z has one of the two following forms:

$$up ::= z :\sim c \,|\, z :\sim y + d$$
$$\text{where } c \in \mathbb{Q}_+,\, d \in \mathbb{Q},\, y \in \mathbb{X} \text{ and } \sim\, \in \{<, \leq, =, \neq, \geq, >\}$$

Let v be a valuation and up be a simple update over z. A valuation v' is in $up(v)$ if $v'(y) = v(y)$ for any clock $y \neq z$ and if $v'(z)$ verifies:

$$\begin{cases} v'(z) \sim c & \text{if } up = z :\sim c \\ v'(z) \sim v(y) + d & \text{if } up = z :\sim y + d \end{cases}$$

A *local update* over a set of clocks X is a collection $up = (up_i)_{1 \le i \le k}$ of simple updates, where each up_i is a simple update over some clock $x_i \in X$ (note that it could happen that $x_i = x_j$ for some $i \neq j$). Let $v, v' \in \mathbb{T}^n$ be two clock valuations. We have $v' \in up(v)$ if and only if, for any i, the clock valuation v'' defined by

$$\begin{cases} v''(x_i) = v'(x_i) \\ v''(y) \;\;= v(y) \qquad \text{for any } y \neq x_i \end{cases}$$

verifies $v'' \in up_i(v)$. The terminology *local* comes from the fact that $v'(x)$ depends on x only and not on the other values $v'(y)$.

Example 1. If we take the local update $(x :> y, x :< 7)$, then it means that the value $v'(x)$ must verify : $v'(x) > v(y) \wedge v'(x) < 7$. Note that $up(v)$ may be empty. For instance, the local update $(x :< 1, x :> 1)$ leads to an empty set.

For any subset X of $\mathbb{X}$, $\mathcal{U}(X)$ is the set of local updates which are collections of simple updates over clocks of X. In the following, we need to distinguish the following subsets of $\mathcal{U}(X)$:

- $\mathcal{U}_0(X)$ is the set of reset updates. A reset update up is an update such that for every clock valuations v, v' with $v' \in up(v)$ and any clock $x \in X$, either $v'(x) = v(x)$ or $v'(x) = 0$.
- $\mathcal{U}_{cst}(X)$ is the set of constant updates. A constant update up is an update such that for every clock valuations v, v' with $v' \in up(v)$ and any clock $x \in X$, either $v'(x) = v(x)$ or $v'(x)$ is a rational constant independent of $v(x)$.

2.4 Updatable timed automata

An *updatable timed automaton* over $\mathbb{T}$ is a tuple $\mathcal{A} = (\Sigma, Q, T, I, F, R, X)$, where Σ is a finite alphabet of actions, Q is a finite set of states, $X \subseteq \mathbb{X}$ is a finite set of clocks, $T \subseteq Q \times [\mathcal{C}(X) \times \Sigma \times \mathcal{U}(X)] \times Q$ is a finite set of transitions, $I \subseteq Q$ is the subset of initial states, $F \subseteq Q$ is the subset of final states, $R \subseteq Q$ is the subset of repeated states.
Let $\mathcal{C} \subseteq \mathcal{C}(\mathbb{X})$ be a subset of clock constraints and $\mathcal{U} \subseteq \mathcal{U}(\mathbb{X})$ be a subset of updates, the class $Aut(\mathcal{C},\mathcal{U})$ is the set of all timed automata whose transitions only use clock constraints of $\mathcal{C}$ and updates of $\mathcal{U}$. The usual class of timed automata, defined in [AD90], is the family $Aut(\mathcal{C}_{df}(\mathbb{X}),\mathcal{U}_0(\mathbb{X}))$.
A *path* in $\mathcal{A}$ is a finite or an infinite sequence of consecutive transitions:

$$P = q_0 \xrightarrow{\varphi_1, a_1, up_1} q_1 \xrightarrow{\varphi_2, a_2, up_2} q_2 \ldots, \text{ where } (q_{i-1}, \varphi_i, a_i, up_i, q_i) \in T,\ \forall i > 0$$

The path is said to be *accepting* if it starts in an initial state ($q_0 \in I$) and *either* it is finite and it ends in an final state, *or* it is infinite and passes infinitely often through a repeated state. A *run* of the automaton through the path P is a sequence of the form:

$$\langle q_0, v_0 \rangle \xrightarrow[t_1]{\varphi_1, a_1, up_1} \langle q_1, v_1 \rangle \xrightarrow[t_2]{\varphi_2, a_2, up_2} \langle q_2, v_2 \rangle \ldots$$

where $\tau = (t_i)_{i \geq 1}$ is a time sequence and $(v_i)_{i \geq 0}$ are clock valuations such that:

$$\begin{cases} v_0(x) = 0, \; \forall x \in \mathbb{X} \\ v_{i-1} + (t_i - t_{i-1}) \models \varphi_i \\ v_i \in up_i\,(v_{i-1} + (t_i - t_{i-1})) \end{cases}$$

Remark that any set $up_i(v_{i-1} + (t_i - t_{i-1}))$ of a run is non empty.
The label of the run is the timed word $w = (a_1, t_1)(a_2, t_2)\ldots$ If the path P is accepting then the timed word w is said to be accepted by the timed automaton. The set of all timed words accepted by $\mathcal{A}$ over the time domain $\mathbb{T}$ is denoted by $L(\mathcal{A}, \mathbb{T})$, or simply $L(\mathcal{A})$.

Remark 1. A "folklore" result on timed automata states that the families $Aut(\mathcal{C}(\mathbb{X}),\mathcal{U}_0(\mathbb{X}))$ and $Aut(\mathcal{C}_{df}(\mathbb{X}),\mathcal{U}_0(\mathbb{X}))$ are language-equivalent. This is because any classical timed automaton (using reset updates only) can be transformed into a diagonal-free classical timed automaton recognizing the same language (see [BDGP98] for a proof). Another "folklore" result states that constant updates are not more powerful than reset updates *i.e.* the families $Aut(\mathcal{C}(\mathbb{X}),\mathcal{U}_{cst}(\mathbb{X}))$ and $Aut(\mathcal{C}(\mathbb{X}),\mathcal{U}_0(\mathbb{X}))$ are language-equivalent.

3 The Emptiness Problem

For verification purposes, a fundamental question about timed automata is to decide whether the accepted language is empty. This problem is called the *emptiness problem.* To simplify, we will say that a class of timed automata is *decidable* if the emptiness problem is decidable for this class. The following result, due to Alur and Dill [AD90], is one of the most important about timed automata.

Theorem 1. *The class $Aut(\mathcal{C}(\mathbb{X}),\mathcal{U}_0(\mathbb{X}))$ is decidable.*

The principle of the proof is the following. Let $\mathcal{A}$ be an automaton of $Aut(\mathcal{C}(\mathbb{X}),\mathcal{U}_0(\mathbb{X}))$, then a Büchi automaton (often called the *region automaton* of $\mathcal{A}$) which recognizes the *untimed language* Untime($L(\mathcal{A})$) of $L(\mathcal{A})$ is effectively constructible. The untime language of $\mathcal{A}$ is defined as follows : Untime($L(\mathcal{A})$) $=$ $\{\sigma \in \Sigma^\infty \mid$ there exists a time sequence τ such that $(\sigma, \tau) \in L(\mathcal{A})\}$.
The emptiness of $L(\mathcal{A})$ is obviously equivalent to the emptiness of Untime($L(\mathcal{A})$) and since the emptiness of a Büchi automaton on words is decidable [HU79], the result follows. In fact, the result is more precise: testing emptiness of a timed automaton is Pspace-complete (see [AD94] for the proofs).

Remark 2. From [AD94] (Lemma 4.1) it suffices to prove the theorem above for timed automata where all constants appearing in clock constraints are integers (and not arbitrary rationals). Indeed, for any timed automaton $\mathcal{A}$, there exists some positive integer δ such that for any constant c of a clock constraint of $\mathcal{A}$, $\delta.c$ is an integer. Let $\mathcal{A}'$ be the timed automaton obtained from $\mathcal{A}$ by replacing each constant c by $\delta \cdot c$, then it is immediate to verify that $L(\mathcal{A}')$ is empty if and only if $L(\mathcal{A})$ is empty.

4 Undecidable Classes of Updatable Timed Automata

In this section we exhibit some important classes of updatable timed automata which are undecidable. All the proofs are reductions of the emptiness problem for counter machines.

4.1 Two counters machine

Recall that a two counters machine is a finite set of instructions over two counters (x and y). There are two types of instructions over counters:

- *incrementation instruction* of counter $i \in \{x, y\}$:

$$p : i := i + 1\,;\ \text{goto}\ q\ \ \text{(where } p \text{ and } q \text{ are instruction labels)}$$

- *decrementation (or zero-testing) instruction* of counter $i \in \{x, y\}$:

$$p :\ \text{if}\ i > 0\ \begin{cases} \texttt{then}\ i := i - 1\,;\ \text{goto}\ q \\ \texttt{else}\ \ \text{goto}\ q' \end{cases}$$

The machine starts at instruction labelled by s_0 with $x = y = 0$ and stops at a special instruction Halt labelled by s_f.

Theorem 2. *The emptiness problem of two counters machine is undecidable [Min67].*

4.2 Diagonal-free automata with updates $x := x - 1$

We consider here a diagonal-free constraints class.

Proposition 1. *Let $\mathcal{U}$ be a set of updates containing both $\{x := x - 1 \mid x \in \mathbb{X}\}$ and $\mathcal{U}_0(\mathbb{X})$. Then the class $Aut(\mathcal{C}_{df}(\mathbb{X}), \mathcal{U})$ is undecidable.*

Sketch of proof. We simulate a two counters machine $\mathcal{M}$ with an updatable timed automaton $\mathcal{A}_{\mathcal{M}} = (\Sigma, Q, T, I, F, R, X)$ with $X = \{x, y, z\}$, $\Sigma = \{a\}$ (for convenience reasons labels are omitted in the proof) and equipped with updates $x := x - 1$ and $y := y - 1$. Clocks x and y simulate the two counters.

Simulation of an increment appears on Figure 1. Counter x is implicitly incremented by letting the time run during 1 unit of time (this is controlled with the test $z = 1$). Then the other counter y is decremented with the $y := y - 1$ update.

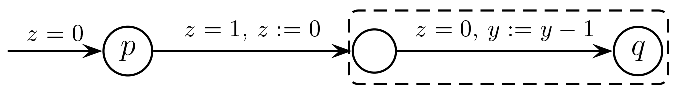


Fig. 1. *Simulation of a incrementation operation over counter x.*

Simulation of a decrement appears on Figure 2. Counter x is either decremented using the $x := x - 1$ update if $x \geq 1$, or unchanged otherwise.

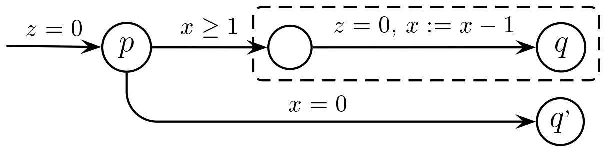


Fig. 2. *Simulation of a decrementation operation on the counter x.*

Remark that we never compare two clocks but only use guards of the form $i \sim c$ with $i \in \{x, y, z\}$ and $c \in \{0, 1\}$.
To complete the definition of $\mathcal{A}_{\mathcal{M}}$, we set $I = \{s_0\}$ and $F = \{s_f\}$. The language of $\mathcal{M}$ is empty if and only if the language of $\mathcal{A}_{\mathcal{M}}$ is empty and this implies undecidability of emptiness problem for the class $Aut(\mathcal{C}_{df}(\mathbb{X}), \mathcal{U})$.

4.3 Automata with updates $x := x + 1$ or $x :> 0$ or $x :> y$ or $x :< y$

Surprisingly, classes of arbitrary timed automata with special updates are undecidable.

Proposition 2. *Let $\mathcal{U}$ be a set of updates containing $\mathcal{U}_0(\mathbb{X})$ and* (**1**) $\{x := x + 1 \mid x \in \mathbb{X}\}$ *or* (**2**) $\{x :> 0 \mid x \in \mathbb{X}\}$ *or* (**3**) $\{x :> y \mid x, y \in \mathbb{X}\}$ *or* (**4**) $\{x :< y \mid x, y \in \mathbb{X}\}$*, then the class $Aut(\mathcal{C}(\mathbb{X}), \mathcal{U})$ is undecidable.*

Sketch of proof. The proofs are four variations of the construction given for proposition 1. The idea is to replace every transition labelled with updates $x := x - 1$ or $y := y - 1$ (framed with dashed lines on pictures) by a small automaton involving the other kinds of updates only. The counter machine will be now simulated by an updatable timed automaton with four clocks $\{w, x, y, z\}$. We show how to simulate an $x := x - 1$ in any of the four cases :

(**1**) Firstly clock w is reset, then update $w := w + 1$ is performed until $x - w = 1$ (recall that x simulates a counter and that we are interested to its integer values). Secondly, clock x is reset and update $x := x + 1$ is performed until $x = w$.
(**2**) A $w :> 0$ is guessed, followed by a test $x - w = 1$. Then a $x :> 0$ is guessed, followed by a test $x = w$.
(**3**) Clock w is reset, $w :> w$ is guessed and test $x - w = 1$ is made. Then clock x is reset, $x :> x$ is guessed and test $x = w$ is made.
(**4**) A $w :< x$ is guessed, followed by test $x - w = 1$. Then a $x :< x$ is guessed, followed by a test $x = w$.

In the four cases, operations are made instantaneously with the help of test $z = 0$ performed at the beginning and at the end of the decrementation simulation. Remark that for any case we use comparisons of clocks. We will see in section 6 that classes of diagonal-free timed automata equipped with any of these four updates are decidable.

Let us end the current section with a result about *mixed updates*. Updates of the kind $y + c \leq: x :\leq z + d$ (with c, $d \in \mathbb{N}$) can simulate clock comparisons. In fact, in order to simulate a test $x - w = 1$, it suffices to guess a $w + 1 \leq: z' :\leq x$

followed by a $x \leq: z' :\leq w+1$. Both guesses have solutions if and only if $[w+1; x] = [x; w+1] = \{x\}$ if and only if $(x - w = 1)$. In conclusion, we cannot mix different kinds of updates anyhow, while keeping diagonal-free automata decidable:

Proposition 3. *Let $\mathcal{U}$ be a set of updates containing $\mathcal{U}_0(\mathbb{X})$ and $\{x + c \leq: y :\leq z + d \,|\, x, y, z \in \mathbb{X},\ c, c' \in \mathbb{N}\}$. Then the class $Aut(\mathcal{C}_{df}(\mathbb{X}), \mathcal{U})$ is undecidable.*

5 Construction of an Abstract Region Automaton

We want to check emptiness of the timed language accepted by some timed automaton. To this aim, we will use a technique based on the original construction of the region automaton ([AD94]).

5.1 Construction of a region graph

Let $X \subset \mathbb{X}$ be a finite set of clocks. A *family of regions* over X is a couple $(\mathcal{R}, \mathtt{Succ})$ where $\mathcal{R}$ is a finite set of regions (*i.e.* of subsets of $\mathbb{T}^X$) and the *successor function* $\mathtt{Succ} : \mathcal{R} \to \mathcal{R}$ verifies that for any region $R \in \mathcal{R}$ the following holds:

- for each $v \in R$, there exists $t \in \mathbb{T}$ such that $v + t \in \mathtt{Succ}(R)$ and for every $0 \leq t' \leq t$, $v + t' \in (R \cup \mathtt{Succ}(R))$
- if $v \in R$, then for all $t \in \mathbb{T}$, $v + t \in \mathtt{Succ}^*(R)$

Let $\mathcal{U} \subset \mathcal{U}(X)$ be a finite set of updates. Each update $up \in \mathcal{U}$ induces naturally a function $\widehat{up} : \mathcal{R} \to \mathcal{P}(\mathcal{R})$ which maps each region R into the set $\{R' \in \mathcal{R} \,|\, up(R) \cap R' \neq \emptyset\}$. The set of regions $\mathcal{R}$ is *compatible* with $\mathcal{U}$ if for all $up \in \mathcal{U}$ and for all $R,\ R' \in \mathcal{R}$:

$$R' \in \widehat{up}(R) \iff \forall v \in R, \exists v' \in R' \text{ such that } v' \in up(v)$$

Then, the *region graph* associated with $(\mathcal{R}, \mathtt{Succ}, \mathcal{U})$ is a graph whose set of nodes is $\mathcal{R}$ and whose vertices are of two distinct types:

$$\begin{array}{ll} R \longrightarrow R' & \text{if } R' = \mathtt{Succ}(R) \\ R \Longrightarrow_{up} R' & \text{if } R' \in \widehat{up}(R) \end{array}$$

Let $\mathcal{C} \subset \mathcal{C}(X)$ be a finite set of clock constraints. The set of regions $\mathcal{R}$ is *compatible* with $\mathcal{C}$ if for all $\varphi \in \mathcal{C}$ and for all $R \in \mathcal{R}$: either $R \subseteq \varphi$ or $R \subseteq \neg\varphi$.

5.2 Construction of the region automaton

Let $\mathcal{A}$ be a timed automaton in $Aut(\mathcal{C}, \mathcal{U})$. Let $(\mathcal{R}, \mathtt{Succ})$ be a family of regions such that $\mathcal{R}$ is compatible with $\mathcal{C}$ and $\mathcal{U}$. We define the *region automaton* $\Gamma_{\mathcal{R},\mathtt{Succ}}(\mathcal{A})$ associated with $\mathcal{A}$ and $(\mathcal{R}, \mathtt{Succ})$, as the finite (untimed) automaton defined as follows:

- Its set of locations is $Q \times \mathcal{R}$; its initial locations are $(q_0, \mathbf{0})$ where q_0 is initial and $\mathbf{0}$ is the region where all clocks are equal to zero; its repeated locations are (r, R) where r is repeated in $\mathcal{A}$ and R is any region; its final locations are (f, R) where f is final in $\mathcal{A}$ and R is any region.
- Its transitions are defined by:
  · $(q, R) \xrightarrow{\varepsilon} (q, R')$ if $R \to R'$ is a transition of the region graph,
  · $(q, R) \xrightarrow{a} (q', R')$ if there exists a transition (q, φ, a, up, q') in $\mathcal{A}$ such that $R \subseteq \varphi$ and $R \Longrightarrow_{up} R'$ is a transition of the region graph.

Theorem 3. *Let $\mathcal{A}$ be a timed automaton in $Aut(\mathcal{C},\mathcal{U})$ where $\mathcal{C}$ (resp. $\mathcal{U}$) is a finite set of clock constraints (resp. of updates). Let $(\mathcal{R}, \texttt{Succ})$ be a family of regions such that $\mathcal{R}$ is compatible with $\mathcal{C}$ and $\mathcal{U}$. Then the automaton $\Gamma_{\mathcal{R},\texttt{Succ}}(\mathcal{A})$ accepts the language* $\textsc{Untime}(L(\mathcal{A}))$.

Assume we can encode a region in a polynomial space, then we can decide the emptiness of the language in polynomial space. It suffices to guess an accepted run in the automaton by remembering only the two current successive configurations of the region automaton (this is the same proof than in [AD94]).

We will now study some classes of timed automata, and consider particular regions which verify the conditions required by the region automaton. This will lead us to some decidability results using the above construction.

6 Considering Diagonal-Free Updatable Timed Automata

Definition of the regions we consider - We consider a finite set of clocks $X \subset \mathbb{X}$. We associate an integer constant c_x to each clock $x \in X$, and we define the set of intervals:

$$\mathcal{I}_x = \{[c] \mid 0 \leq c \leq c_x\} \cup \{]c; c+1[\mid 0 \leq c < c_x\} \cup \{]c_x; +\infty[\}$$

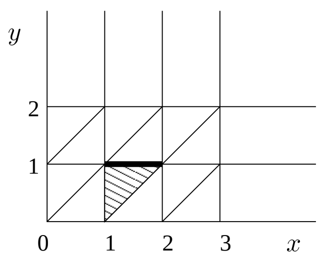
Let α be a tuple $((I_x)_{x \in X}, \prec)$ where:

- $\forall x \in X,\ I_x \in \mathcal{I}_x$
- $\prec$ is a total preorder on $X_0 = \{x \in X \mid I_x \text{ is an interval of the form }]c; c+1[\}$

The *region* (defined by) α is thus

$$R(\alpha) = \left\{ v \in \mathbb{T}^X \;\middle|\; \begin{array}{l} \forall x \in X,\ v(x) \in I_x \\ \forall x, y \in X_0, \text{ the following holds} \\ \quad x \prec y \iff \texttt{frac}(v(x)) \leq \texttt{frac}(v(y)) \end{array} \right\}$$

The set of all regions defined in such a way will be denoted by $\mathcal{R}_{(c_x)_{x \in X}}$.



Example 2. As an example, assume we have only two clocks x and y with the constants $c_x = 3$ and $c_y = 2$. Then, the set of regions associated with those constants is described in the figure beside. The hashed region is defined by the following: $I_x =]1; 2[$, $I_y =]0; 1[$ and the preorder $\prec$ is defined by $x \prec y$ and $y \not\prec x$.

We obtain immediately the following proposition:

Proposition 4. *Let $\mathcal{C} \subseteq \mathcal{C}_{df}(X)$ be such that for any clock constraint $x \sim c$ of $\mathcal{C}$, it holds $c \leq c_x$. Then the set of regions $\mathcal{R}_{(c_x)_{x\in X}}$ is compatible with $\mathcal{C}$.*

Note that the result does not hold for any set of constraints included in $\mathcal{C}(X)$. For example, the region $(]1;+\infty[\times]1;+\infty[,\emptyset)$ is neither included in $x - y \leq 1$ nor in $x - y > 1$.

Computation of the successor function - Let $R = ((I_x)_{x\in X}, \prec)$ be a region. We set $Z = \{x \in X \mid I_x \text{ is of the form } [c]\}$. Then the region $\mathtt{Succ}(R) = ((I'_x)_{x\in X}, \prec')$ is defined as follows, distinguishing two cases:

1. If $Z \neq \emptyset$, then
   - $I'_x = \begin{cases} I_x & \text{if } x \notin Z \\ ]c, c+1[& \text{if } I_x = [c] \text{ with } c \neq c_x \\ ]c_x, \infty[& \text{if } I_x = [c_x] \end{cases}$
   - $x \prec' y$ if $(x \prec y)$ or $I_x = [c]$ with $c \neq c_x$ and I'_y has the form $]d, d+1[$
2. If $Z = \emptyset$, let M be the set of maximal elements of $\prec$. Then
   - $I'_x = \begin{cases} I_x & \text{if } x \notin M \\ [c+1] & \text{if } x \in M \text{ and } I_x =]c, c+1[\end{cases}$
   - $\prec'$ is the restriction of $\prec$ to $\{x \in X \mid I'_x \text{ has the form }]d, d+1[\}$

Taking the previous example, the successor of the gray region is defined by $I_x =]1;2[$ and $I_y = [1]$ (drawn as the thick line).

We will now define a suitable set of updates compatible with the regions.

What about the updates ? - We consider now a **local** update $up = (up_x)_{x\in X}$ over a finite set of clocks $X \subset \mathbb{X}$ such that for any clock x, up_x is in one of the four following subsets of $\mathcal{U}(X)$, each of them being given by an abstract grammar:

- $det_x ::= x := c \mid x := z + d$ with $c \in \mathbb{N}$, $d \in \mathbb{Z}$ and $z \in X$.
- $inf_x ::= x :\lhd c \mid x :\lhd z + d \mid inf_x \wedge inf_x$ with $\lhd \in \{<, \leq\}$, $c \in \mathbb{N}$, $d \in \mathbb{Z}$ and $z \in X$.
- $sup_x ::= x :\rhd c \mid x :\rhd z + d \mid sup_x \wedge sup_x$ with $\rhd \in \{>, \geq\}$, $c \in \mathbb{N}$, $d \in \mathbb{Z}$ and $z \in X$.
- $int_x ::= x :\in (c; d) \mid x :\in (c; z + d) \mid x :\in (z + c; d) \mid x :\in (z + c; z + d)$ where (and) are either [or], z is a clock and c, d are in $\mathbb{Z}$.

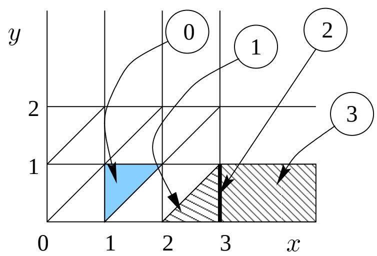
Let us denote by $\mathcal{U}_1(X)$ this set of local updates. As in the case of simple updates, we will give a necessary and sufficient condition for R' to be in $\widehat{up}(R)$ when R, R' are regions and up is a local update.

Case of simple updates - We will first prove that for any **simple** update up, $\mathcal{R}_{(c_x)_{x\in X}}$ is compatible with up. To this aim, we construct the regions belonging to $\widehat{up}(R)$ by giving a necessary and sufficient condition for a given region R' to be in $\widehat{up}(R)$.
Assume that $R = ((I_x)_{x\in X}, \prec)$ where $\prec$ is a total preorder on X_0 and that up is a simple update over z, then the region $R' = ((I'_x)_{x\in X}, \prec')$ (where $\prec'$ is a total preorder on X'_0) is in $\widehat{up}(R)$ if and only if $I'_x = I_x$ for all $x \neq z$ and :

if $\mathbf{up = z :\sim c}$ with $\mathbf{c} \in \mathbb{N}$: I'_z can be any interval of $\mathcal{I}_z$ which intersects $\{\gamma \mid \gamma \sim c\}$ and
- either I'_z has the form $[d]$ or $]c_z;+\infty[$, $X'_0 = X_0 \setminus \{z\}$ and $\prec' = \prec \cap (X'_0 \times X'_0)$.
- either I'_z has the form $]d;d+1[$, $X'_0 = X_0 \cup \{z\}$ and $\prec'$ is any total preorder which coincides with $\prec$ on $X_0 \setminus \{z\}$.

if $\mathbf{up = z :\sim y + c}$ with $\mathbf{c} \in \mathbb{Z}$: we assume in this case that $c_z \leq c_y + c$. Thus if I_y is any interval in $\mathcal{I}_y$ then $I_y + c$ is included in an interval of $\mathcal{I}_z$ (in particular, whenever I_y is non bounded then $I_y + c$ is non bounded, which is essential in order to prove the compatibility).
I'_z can be any interval of $\mathcal{I}_z$ such that there exists $\alpha \in I'_z$, $\beta \in I_y$ with $\alpha \sim \beta + c$ and
- either I'_z has the form $[d]$ or $]c_z;+\infty[$, $X'_0 = X_0 \setminus \{z\}$ and $\prec' = \prec \cap (X'_0 \times X'_0)$.
- either I'_z has the form $]d;d+1[$, $X'_0 = X_0 \cup \{z\}$ and
  - If $y \notin X_0$, $\prec'$ is any total preorder on X'_0 which coincides with $\prec$ on $X_0 \setminus \{z\}$.
  - If $y \in X_0$, then:
    - either $I_y + c \neq I'_z$ and $\prec'$ is any total preorder on X'_0 which coincides with $\prec$ on $X_0 \setminus \{z\}$
    - either $I_y + c = I'_z$ and $\prec'$ is any total preorder on X'_0 which coincides with $\prec$ in $X_0 \setminus \{z\}$ and verifies:
      - · $z \prec' y$ and $y \prec' z$ if $\sim$ is $=$
      - · $z \prec' y$ and $y \not\prec' z$ if $\sim$ is $<$
      - · $z \prec' y$ if $\sim$ is $\leq$
      - · $y \prec' z$ if $\sim$ is $\geq$
      - · $z \not\prec' y$ and $y \prec' t$ if $\sim$ is $>$
      - · $(z \prec' y$ and $y \not\prec' z)$ or $(z \not\prec' y$ and $y \prec' z)$ if $\sim$ is $\neq$

From this construction, it is easy to verify that $\mathcal{R}_{(c_x)_{x\in X}}$ is compatible with any simple update.



Example 3. We take the regions described in the figure beside. We want to compute the updating successors of the region 0 by the update $x :> y+2$. The three updating successors are drawn in the figure beside. Their equations are:
- Region 1: $I'_x =]2;3[$, $I'_y =]0;1[$ and $y \prec' x$
- Region 2: $I'_x = [3]$, $I'_y =]0;1[$
- Region 3: $I'_x =]3;+\infty[$, $I'_y =]0;1[$

Remark 3. Note that the fact that updates of the form $z := z-1$ (even used with diagonal-free constraints only) lead to undecidability of emptiness (Section 4), is not in contradiction with our construction. This is because we can not assume that $c_z \leq c_z - 1$.

Case of local updates - We will use the semantics of the local updates from section 2.3 to compute the updating successors of a region. Assume that $R = ((I_x)_{x\in X}, \prec)$ and that $up = (up_x)_{x\in X}$ is a local update over X then $R' = ((I'_x)_{x\in X}, \prec') \in \widehat{up}(R)$ if and only if there exists a total preorder $\prec''$ on a subset of $X \cup X'$ (where X' is a disjoint copy of X) verifying

$$\begin{array}{lcl} y \prec'' z & \iff & y \prec z \text{ for all } y,\, z \in X \\ y' \prec'' z' & \iff & y \prec' z \text{ for all } y,\, z \in X \end{array}$$

and such that, for any simple update up_i appearing in up_x, the region $R_i = ((I_{i,x})_{x\in X}, \prec_i)$ defined by

$$I_{i,x} = \begin{cases} I_x & \text{if } x \neq x_i \\ I'_x & \text{otherwise} \end{cases} \qquad \text{and} \qquad \begin{array}{lcl} \cdot\ y \prec_i z & \iff & y \prec z \quad \text{for } y,\, z \neq x_i \\ \cdot\ x_i \prec_i z & \iff & x'_i \prec'' z \text{ for } z \neq x_i \\ \cdot\ z \prec_i x_i & \iff & z \prec'' x'_i \text{ for } z \neq x_i \end{array}$$

belongs to $\widehat{up}_i(R)$.

Assume now that $\mathcal{U}$ is a set of updates included in $\mathcal{U}_1(X)$. It is then technical, but without difficulties, to show that under the following hypothesis:

- for each simple update $y :\sim z + c$ which is part of some local update of $\mathcal{U}$, condition $c_y \leq c_z + c$ holds

the family of regions $(\mathcal{R}_{(c_x)_{x\in X}}, \texttt{Succ})$ is compatible with $\mathcal{U}$. In fact, the set $X \cup X'$ and the preorder $\prec''$ both encode the original and the updating regions. This construction allows us to obtain the desired result for local updates.

Remark 4. In our definition of $\mathcal{U}_1(X)$, we considered restricted set of local updates. Without such a restriction, it can happen that no such preorder $\prec''$ exists. For example, let us take the local update $x :> y \wedge x :< z$ and the region R defined by $I_x = [0]$, $I_y = I_z =]0;1[$, $z \prec y$ and $y \not\prec z$. Then the preorder $\prec''$ should verify the following : $y \prec'' x'$, $x' \prec'' z$, $z \prec'' y$ and $y \not\prec z$, but this leads to a contradiction. There is no such problem for the local updates from $\mathcal{U}_1(X)$, as we only impose to each clock x' to have a value greater than or lower than some other clock values.

For the while, we have only considered updates with integer constants but an immediate generalization of Remark 2 allows to treat updates with any rational constants. We have therefore proved the following theorem:

Theorem 4. *Let $\mathcal{C} \subseteq \mathcal{C}_{df}(X)$ be a set of diagonal-free clock constraints. Let $\mathcal{U} \subseteq \mathcal{U}_1(X)$ be a set of updates. Let $(c_x)_{x\in X}$ be a family of constants such that for each clock constraint $y \sim c$ of $\mathcal{C}$, condition $c \leq c_y$ holds and for each update $z :\sim y + c$ of $\mathcal{U}$, condition $c_z \leq c_y + c$ holds. Then the family of regions $(\mathcal{R}_{(c_x)_{x\in X}}, \texttt{Succ})$ is compatible with $\mathcal{C}$ and $\mathcal{U}$.*

Remark 5. Obviously, it is not always the case that there exists a family of integer constants such that for each update $y :\sim z + c$ of $\mathcal{U}$, condition $c_y \leq c_z + c$ holds. Nevertheless:

– It is the case when all the constants c appearing in updates $y :\sim z + c$ are non-negative.
– In the general case, the existence of such a family is decidable thanks to results on systems on linear Diophantine inequations [Dom91].

For any couple $(\mathcal{C}, \mathcal{U})$ verifying the hypotheses of theorem 4, by applying theorem 3, the family $Aut(\mathcal{C}, \mathcal{U})$ is decidable. Moreover, since we can encode a region in polynomial space, testing emptiness is PSPACE, and even PSPACE-complete (since it is the case for classical timed automata).

Remark 6. The p-automata used in [BF99] to modelize the ABR protocol can be easily transformed into updatable timed automata from a class which fulfills the hypotheses of theorem 4. Their emptiness is then decidable.

7 Considering Arbitrary Updatable Timed Automata

In this section, we allow arbitrary clock constraints. We thus need to define a bit more complicated set of regions. To this purpose we consider for each pair y, z of clocks (taken in $X \subset \mathbb{X}$ a finite set of clocks), two constants $d^-_{y,z} \leq d^+_{y,z}$ and we define

$$\begin{aligned}\mathcal{J}_{y,z} = \{]-\infty; d^-_{y,z}[\} &\cup \{[d] \mid d^-_{y,z} \leq d \leq d^+_{y,z}\} \cup \\ &\{]d; d+1[\mid d^-_{y,z} \leq d < d^+_{y,z}\} \cup \{]d^+_{y,z}; +\infty[\}\end{aligned}$$

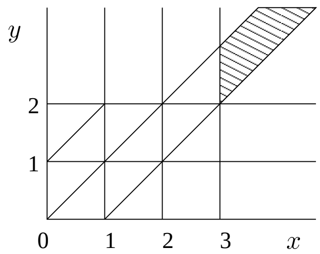
The region defined by a tuple $((I_x)_{x \in X}, (J_{x,y})_{x,y \in X}, \prec)$ where

- $\forall x \in X,\ I_x \in \mathcal{I}_x$
- if $\mathcal{X}_\infty$ denotes the set $\{(y,z) \in X^2 \mid I_y \text{ or } I_z \text{ is non bounded}\}$, then $\forall (y,z) \in \mathcal{X}_\infty,\ J_{y,z} \in \mathcal{J}_{y,z}$
- $\prec$ is a total preorder on $X_0 = \{x \in X \mid I_x \text{ is an interval of the form }]c, c+1[\}$

is the following subset of $\mathbb{T}^X$:

$$\left\{ v \in \mathbb{T}^X \;\middle|\; \begin{array}{l} \forall x \in X,\ v(x) \in I_x \\ \forall x, y \in X_0, \text{ it holds} \\ \quad x \prec y \iff \mathtt{frac}(v(x)) \leq \mathtt{frac}(v(y)) \\ \forall y, z \in \mathcal{X}_\infty,\ v(y) - v(z) \in J_{y,z} \end{array} \right\}$$

In fact, we do not have to keep in mind the values $d^-_{*,*}$ as y and z play symmetrical roles and $d^-_{y,z}$ is equal to $-d^+_{z,y}$, thus we set $d_{y,z} = d^+_{y,z}$. The set of all regions defined in such a way will be denoted by $\mathcal{R}_{(c_y)_{x\in X},(d_{y,z})_{y,z\in X}}$.

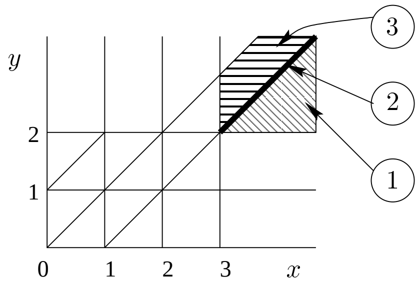


Example 4. Assume that we have only two clocks x and y and that the maximal constants are $c_x = 3$ and $c_y = 2$, with clocks constraints $x - y \sim 0$ and $x - y \sim 1$. Then, the set of regions associated with those constants is described in the figure beside. The gray region is defined by $I_x =]3;+\infty[$, $I_y =]2;+\infty[$ and $-1 < y - x < 0$ (*i.e.* $J_{y,x}$ is $]-1;0[$).

The region $\texttt{Succ}(R)$ can be defined in a way similar to the one used in the diagonal-free case. We also have to notice that this set of regions is compatible with the clock constraints we consider.

Indeed we define the set $\mathcal{U}_2(X)$ of local updates $up = (up_x)_{x\in X}$ where for each clock x, up_x is one of the following simple updates:

$$x := c \mid x := y \mid x :< c \mid x :\leq c$$



From the undecidability results of Section 4, we have to restrict the used updates if we want to preserve decidability. For example, if we consider the update $y := y + 1$ and the regions described in the figure beside, the images of the region 1 are the regions 1, 2 and 3. But we can not reach region 1 (resp. 2, resp. 3) from any point of region 1. Thus, this set of regions does not seem to be compatible with the update $y := y + 1$.

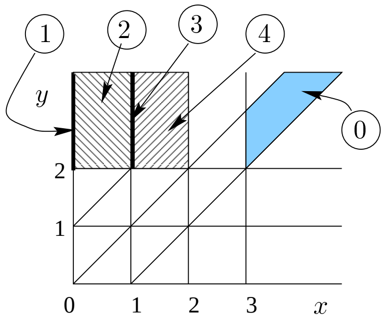
By constructions similar to the ones of Section 6, we obtain the following theorem:

Theorem 5. *Let $\mathcal{C} \subseteq \mathcal{C}(X)$ be a set of clock constraints. Let $\mathcal{U} \subseteq \mathcal{U}_2(X)$ be a set of updates. Let $(c_x)_{x\in X}$ and $(d_{y,z})_{y,z\in X}$ be families of constants such that*

- *for each clock constraint $y \sim c$ of $\mathcal{C}$, condition $c \leq c_y$ holds,*
- *for each clock constraint $x - y \sim c$, condition $c \leq d_{x,y}$ holds,*
- *for each update $y :< c$ or $y :\leq c$ or $y := c$, it holds $c \leq c_y$, and for each clock z, condition $c_z \geq c + d_{y,z}$ holds,*
- *for each update $y := z$, condition $c_y \leq c_z$ holds*

Then the family of regions $(\mathcal{R}_{(c_x)_{x\in X},(d_{y,z})_{y,z\in X}}, \texttt{Succ})$ is compatible with $\mathcal{C}$ and $\mathcal{U}$.

Thus, the class $Aut(\mathcal{C},\mathcal{U})$ is decidable, and as in the previous case, testing emptiness of updatable timed automata is PSPACE-complete (unlike the case of diagonal-free updates, the previous system of Diophantine equations always has a solution).



Example 5. We take the regions we used before. We want to compute the updating successors of the region 0 by the update $x :< 2$. The four updating successors are drawn in the figure beside. Their equations are:
- Region 1: $I'_x = [0]$ and $I'_y =]2; +\infty[$
- Region 2: $I'_x =]0; 1[$, $I'_y =]2; +\infty[$ and $J_{y,x} =]1; +\infty[$
- Region 3: $I'_x = [1]$ and $I'_y =]2; +\infty[$
- Region 4: $I'_x =]1; 2[$, $I'_y =]2; +\infty[$ and $J_{y,x} =]1; +\infty[$

8 Conclusion

The main results of this paper about the emptiness problem are summarized in the following table:

$\mathcal{U}_0(\mathbb{X}) \cup \cdots$	$\mathcal{C}_{df}(\mathbb{X})$	$\mathcal{C}(\mathbb{X})$
$\emptyset$	PSPACE	PSPACE
$\{x := c \mid x \in \mathbb{X}\} \cup \{x := y \mid x,\, y \in \mathbb{X}\}$	PSPACE	PSPACE
$\{x :< c \mid x \in \mathbb{X},\, c \in \mathbb{Q}^+\}$	PSPACE	PSPACE
$\{x := x + 1 \mid x \in \mathbb{X}\}$	PSPACE	Undecidable
$\{x :> c \mid x \in \mathbb{X},\, c \in \mathbb{Q}^+\}$	PSPACE	Undecidable
$\{x :> y \mid x,\, y \in \mathbb{X}\}$	PSPACE	Undecidable
$\{x :< y \mid x,\, y \in \mathbb{X}\}$	PSPACE	Undecidable
$\{x :\sim y + c \mid x,\, y \in \mathbb{X},\, c \in \mathbb{Q}^+\}$	PSPACE	Undecidable
$\{x := x - 1 \mid x \in \mathbb{X}\}$	Undecidable	Undecidable

One of the surprising facts of our study is that the frontier between what is decidable and not depends on the diagonal constraints (except for the $x := x - 1$ update), whereas it is well-known that diagonal constraints do not increase the expressive power of classical timed automata.
Note that, as mentionned before, the decidable classes are not more powerful than classical timed automata in the sense that for any updatable timed automaton of such a class, a classical timed automaton (with $\varepsilon-$transitions) recognizing the same language – and even most often bisimilar – can be effectively constructed [BDFP00b]. However, in most cases an exponential blow-up seems unavoidable. This means that transforming updatable timed automata into classical timed automata cannot constitute an efficient strategy to solve the emptiness problem.

In the existing model-checkers, time is represented through data structures like DBM (Difference Bounded Matrix) or CDD (Clock Difference Diagrams). An interesting and natural question is to study how such structures can be used to deal with updatable timed automata.

Acknowledgements: We thank Béatrice Bérard for helpful discussions.

References

[ACD+92] R. Alur, C. Courcoubetis, D.L. Dill, N. Halbwachs, and H. Wong-Toi. Minimization of timed transition systems. In *Proc. of CONCUR'92*, LNCS 630, 1992.

[ACH94] R. Alur, C. Courcoubetis, and T.A. Henzinger. The observational power of clocks. In *Proc. of CONCUR'94*, LNCS 836, pages 162–177, 1994.

[AD90] R. Alur and D.L. Dill. Automata for modeling real-time systems. In *Proc. of ICALP'90*, LNCS 443, pages 322–335, 1990.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theorical Computer Science*, 126:183–235, 1994.

[AFH94] R. Alur, L Fix, and T.A. Henzinger. A determinizable class of timed automata. In *Proc. of CAV'94*, LNCS 818, pages 1–13, 1994.

[AHV93] R. Alur, T.A. Henzinger, and M. Vardi. Parametric real-time reasoning. In *Proc. of ACM Symposium on Theory of Computing*, pages 592–601, 1993.

[BD00] B. Bérard and C. Dufourd. Timed automata and additive clock constraints. To appear in IPL, 2000.

[BDFP00a] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable ? Research Report LSV-00-3, LSV, ENS de Cachan, 2000.

[BDFP00b] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Expressiveness of updatable timed automata. Research report, LSV, ENS de Cachan, 2000. Submitted to MFCS'2000.

[BDGP98] B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, pages 145–182, 1998.

[BF99] B. Bérard and L. Fribourg. Automatic verification of a parametric real-time program : the ABR conformance protocol. In *Proc. of CAV'99*, LNCS 1633, 1999.

[BFKM99] B. Bérard, L. Fribourg, F. Klay, and J.F. Monin. A compared study of two correctness proofs for the standardized algorithm of ABR conformance. Research Report LSV-99-7, LSV, ENS de Cachan, 1999.

[Dom91] E. Domenjoud. Solving systems of linear diophantine equations : an algebraic approach. In *Proc. of MFCS'91*, LNCS 520, pages 141–150, 1991.

[HHWT95] T.A. Henzinger, P. Ho, and H. Wong-Toi. A user guide to HyTech. In *Proc. of TACAS'95*, LNCS 1019, pages 41–71, 1995.

[HHWT97] T.A. Henzinger, P. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *Software Tools for Technology Transfer*, pages 110–122, 1997.

[HKWT95] T.A. Henzinger, P.W. Kopke, and H. Wong-Ti. The expressive power of clocks. In *Proc. of ICALP'95*, LNCS 944, pages 335–346, 1995.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages and computation.* Addison Wesley, 1979.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1:134–152, 1997.

[Min67] M. Minsky. *Computation: finite and infinite machines.* Prentice Hall Int., 1967.

[Wil94] T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 863, 1994.

[Yov97] S. Yovine. A verification tool for real-time systems. *Springer International Journal of Software Tools for Technology Transfer*, 1, October 1997.