Updatable Timed Automata ⋆

Patricia Bouyer[1], Catherine Dufourd[2], Emmanuel Fleury[3], and Antoine Petit[1]

[1] LSV, CNRS UMR 8643 & ENS de Cachan,
61, Av. du Président Wilson,
94235 Cachan Cedex, France
Email: `{bouyer,petit}@lsv.ens-cachan.fr`
[2] EDF – R&D – Dépt. OSIRIS – 1, Av. du Général de Gaulle
92141 Clamart Cedex, France
Email: `catherine.dufourd@edf.fr`
[3] BRICS⋆⋆, Aalborg University
Fredrik Bajers Vej 7
9220 Aalborg Ø, Denmark
Email: `fleury@cs.auc.dk`

Abstract. We investigate extensions of Alur and Dill's timed automata, based on the possibility to update the clocks in a more elaborate way than simply reset them to zero. We call these automata *updatable timed automata*. They form an undecidable class of models, in the sense that emptiness checking is not decidable. However, using an extension of the region graph construction, we exhibit interesting decidable subclasses. In a surprising way, decidability depends on the nature of the clock constraints which are used, diagonal-free or not, whereas these constraints play identical roles in timed automata. We thus describe in a quite precise way the thin frontier between decidable and undecidable classes of updatable timed automata.

We also study the expressive power of updatable timed automata. It turns out that any updatable automaton belonging to some decidable subclass can be effectively transformed into an equivalent timed automaton without updates but with silent transitions. The transformation suffers from an enormous combinatorics blow-up which seems unavoidable. Therefore, updatable timed automata appear to be a concise model for representing and analyzing large classes of timed systems.

1 Introduction

Since their introduction by Alur and Dill [AD90,AD94], timed automata are one of the most-studied and most-established models for real-time systems. Numerous works have been devoted to the "theoretical" comprehension of timed automata (among them, see [ACD+92], [AHV93], [AFH94], [ACH94], [Wil94], [HKWT95]). However the major property of timed automata is probably that emptiness checking is a decidable problem for this model [AD94]. Based on this nice theoretical result, several model-checkers have been developed (for instance CMC[1] [LL98], HyTech[2] [HHWT95,HHWT97], Kronos[3] [Yov97] and Uppaal[4] [LPY97,BLL+98]) and a lot of case studies have been treated (see the web pages of the tools).

⋆ This work has been partly supported by the french RNTL project "Averroes" and french-indian CEPIPRA project n°2102 − 1.

⋆⋆ Basic Research in Computer Science (`http://www.brics.dk`), funded by the Danish National Research Foundation.

[1] `http://www.lsv.ens-cachan.fr/~fl/cmcweb.html`

[2] `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`

[3] `http://www-verimag.imag.fr/TEMPORISE/kronos/`

[4] `http://www.uppaal.com/`

A lot of work has naturally been devoted to extensions of timed automata, with much interest for classes whose emptiness problem remains decidable. There are two main (non exclusive) reasons for extending existing models. First, they can be used to model strictly larger classes of systems and therefore treat more case studies. They also lead to more compact representations of some systems. Conciseness makes modelling easier, in the same way advanced programming languages make the writing of programs easier than with assembly languages.

Considering timed automata, extensions can be obtained in various ways. Recall that in a timed automaton, a transition is guarded by a constraint over a set of variables, called clocks. This constraint has to be satisfied in order to enable the transition. Right after the transition is taken, a subset of clocks is reset to zero. This set of clocks is specified in the label of the transition. The constraints used in Alur and Dill's original model allow to compare (the value of) a clock, or the difference between two clocks, with a rational constant. Note that comparing the sum of two clocks with a constant leads to an undecidable class of automata (see [AD94] but also [Duf97,BD00] where more precise results on the number of clocks are given). Periodic clock constraints, as defined in [CG00], allow to express properties like "the value of a clock is even" or "the value of a clock is of the form $0.5 + 3n$ where n is some integer. The corresponding class of automata is strictly more powerful than Alur and Dill's timed automata if silent transitions (or ε-transitions) are not allowed but coincides with the original model otherwise. Note that, contrary to the untimed setting, silent transitions strictly increase the expressive power of the model (see [BGP96,DGP97] or [BDGP98] for a survey). Several other exotic extensions have been proposed among which we can mention [DZ98] where subsets of clocks can be "freezed".

The aim of the present paper is to investigate an other way to extend the model, with new operations on the clocks. As we recalled just above, in Alur and Dill's model, when a transition is taken, a specified subset of clocks is reset to zero. Our goal is to study more complex updates on clocks, with a particular attention to the decidability of the emptiness problem and to the expressive power of the corresponding classes of automata. We will first study "deterministic" updates where a clock can be reset to a given constant, which does not have to be zero anymore, or to the value of another clock, or more generally to the sum of a constant and of the value of an other clock. We will then be interested in "non-deterministic" updates, where a clock can be reset to an arbitrary value greater than some fixed constant. Note that this type of updates appear sometimes naturally, for example in models of telecommunication protocols (see e.g. the study of the ABR protocol proposed in [BF99,BFKM03]). In the sequel, we will call the corresponding automata, *updatable timed automata*.

It is easy to verify that such updates, even if we only use deterministic ones, lead to an undecidable class of automata. Indeed, it is easy to simulate a two-counter machine (or Minsky machine) with an updatable timed automaton. But it turns out that very interesting subclasses of updatable timed automata can be proven decidable. A surprising result is that decidability often depends on the clock constraints – diagonal-free (*i.e.* where the only allowed comparisons are between a clock and a constant) or not (where differences of two clocks can also be compared with constants). This point makes an important difference with "classical" (*i.e.* Alur and Dill's) timed automata for which it is well-known that these two kinds of constraints have the same expressive power. We show for instance that updates of the form $x := x + 1$ lead to an undecidable class of timed automata if arbitrary clock constraints are allowed but to a decidable class if only diagonal-free clock constraints are allowed. Note that automata with updates of the form $x := x - 1$ always form an undecidable class whatever constraints, diagonal-free or general, are used. We will show that decidability is often not far from undecidability and we will describe in a quite thin way the frontier between the two worlds.

Decidability results are obtained through a generalization of the region graph proposed by Alur and Dill. Given a timed automaton, and using the region graph, a finite automaton can be constructed, which recognizes exactly the untiming of the language recognized by the original timed automaton. Note that the region graph depends on the class of constraints, diagonal-free or not, and on updates. The main difficulty is then to prove that a given set of updates is "compatible" (in a sense which will be of course precisely defined in the paper) with the region graph. This compatibility has to be proven for all updates, not only for resets as was the case in the original model, but also for deterministic and non-deterministic updates as described previously. We will finally see that the complexity of this decision procedure remains PSPACE-complete.

In this paper, we also study the expressive power of updatable timed automata. We show that they are not more powerful than classical timed automata in the sense that for any updatable timed automaton, that belongs to some decidable subclass, a classical timed automaton (potentially with ε-transitions) recognizing the same language – and even most often bisimilar – can be effectively constructed. However in most cases, an exponential blow-up seems unavoidable and thus a transformation into a classical timed automaton does not lead to an efficient decision procedure. This exponential blow-up suggests that we can have much more concise models if using updatable timed automata than if we only use classical timed automata.

The paper is organized as follows. In section 2, we present basic definitions of timed words, clock constraints and updates. Updatable timed automata are defined in section 3 where the emptiness problem is briefly introduced. Section 4 is devoted to our undecidability results. We first reduce an undecidable problem on two counter machines to the emptiness problem for a subclass of updatable timed automata. We then deduce that for several other subclasses of updatable timed automata, emptiness is also undecidable. In section 5, we first propose a generalization of the region automaton principle first described by Alur and Dill. We then use this extension to exhibit large subclasses of updatable timed automata for which emptiness is decidable, when only diagonal-free clock constraints are used (section 5.2) and then when arbitrary clock constraints (section 5.3) are used. The question of the expressive power of updatable timed automata is addressed in section 6. A short conclusion summarizes our results and propose some open questions or developments.

This journal paper is the full version corresponding to the two conference papers [BDFP00a,BDFP00b].

2 Preliminaries

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. We note $Z^\infty = Z^* \cup Z^\omega$. We consider as time domain $\mathbb{T}$ the set $\mathbb{Q}^+$ of non-negative rationals or the set $\mathbb{R}^+$ of non-negative reals and Σ as a finite set of *actions*. A *time sequence* over $\mathbb{T}$ is a finite (or infinite) non decreasing sequence $\tau = (t_i)_{1 \leq i} \in \mathbb{T}^\infty$. A *timed word* $\omega = (a_i, t_i)_{1 \leq i}$ is an element of $(\Sigma \times \mathbb{T})^\infty$, also written as a pair $\omega = (\sigma, \tau)$, where $\sigma = (a_i)_{1 \leq i}$ is a word in Σ^∞ and $\tau = (t_i)_{1 \leq i}$ a time sequence in $\mathbb{T}^\infty$ of same length.
We consider a finite set X of variables, called *clocks*. A *clock valuation* over X is a mapping $v : X \to \mathbb{T}$ that assigns to each clock a time value. The set of all clock valuations over X is denoted $\mathbb{T}^X$. Let $t \in \mathbb{T}$, the valuation $v + t$ is defined by $(v + t)(x) = v(x) + t$, $\forall x \in X$.

2.2 Clock Constraints

Given a set of clocks X, we introduce two sets of clock constraints over X. The most general one, denoted by $\mathcal{C}(X)$, allows to compare a clock or the difference of two clocks with a constant. It is formally defined by the following grammar:

$$\varphi \ ::= \ x \sim c \mid x - y \sim c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$
$$\text{where } x,\ y \in X,\ c \in \mathbb{Q},\ \sim \in \{<, \leq, =, \neq, \geq, >\}$$

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

$$\begin{aligned}\varphi \ ::= \ & x \sim c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi, \\ & \text{where } x \in X,\ c \in \mathbb{Q} \text{ and } \sim \in \{<, \leq, =, \neq, \geq, >\}\end{aligned}$$

Note that this restricted set of constraints is called diagonal-free because constraints of the form $x - y \sim c$ are called *diagonal* clock constraints.

Clock constraints are interpreted over clock valuations. The satisfaction relation, denoted as "$v \models \varphi$" if valuation v satisfies the clock constraint φ, is defined in a natural way for both sets of constraints:

$$\begin{cases} v \models x \sim c & \text{if } v(x) \sim c \\ v \models x - y \sim c & \text{if } v(x) - v(y) \sim c \\ v \models \varphi_1 \wedge \varphi_2 & \text{if } v \models \varphi_1 \text{ and } v \models \varphi_2 \\ v \models \varphi_1 \vee \varphi_2 & \text{if } v \models \varphi_1 \text{ or } v \models \varphi_2 \end{cases}$$

2.3 Updates

Clock constraints allow to test the values of the clocks. In order to change these values, we use the notion of *updates* which are functions from $\mathbb{T}^X$ to $\mathcal{P}(\mathbb{T}^X)$[5]. An update hence associates with each valuation a set of valuations.

In this work, we restrict to a small class of updates, the so-called *local updates*, constructed in the following way. We first define a *simple update* over a clock z as one of the two following functions:

$$\begin{aligned} up \ ::= \ & z :\sim c \mid z :\sim y + d \\ & \text{where } c, d \in \mathbb{Q},\ y \in X \text{ and } \sim \in \{<, \leq, =, \neq, \geq, >\}\end{aligned}$$

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 $v'(z)$ satisfies:

$$\begin{cases} v'(z) \sim c \wedge v'(z) \geq 0 & \text{if } up = z :\sim c \\ v'(z) \sim v(y) + d \wedge v'(z) \geq 0 & \text{if } up = z :\sim y + d \end{cases}$$

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

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

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

Example 1. Let us consider the local update $up = (x :> y, x :< 7)$. Let $v,\ v'$ be two valuations. It holds that $v' \in up(v)$ if $v'(x) > v(y) \wedge v'(x) < 7$.

Note that $up(v)$ may be empty. For instance, the local update $(x :< 1, x :> 1)$ leads to an empty set.

For any set of clocks X, we denote by $\mathcal{U}(X)$ the set of local updates over X. In this paper, we will simply call updates these local updates. The following subsets of $\mathcal{U}(X)$ will play an important role in the rest of the paper.

[5] $\mathcal{P}(\mathbb{T}^X)$ denotes the powerset of $\mathbb{T}^X$.

- $\mathcal{U}_0(X)$ is the set of reset updates. A *reset update* is a local update up such that each simple update defining up is of the form $x := 0$.
- $\mathcal{U}_{\text{cst}}(X)$ is the set of "constant" updates, that is the set of updates up such that each simple update defining up is of the form $x := c$ with $c \in \mathbb{Q}$.
- $\mathcal{U}_{\text{det}}(X)$ is the set of deterministic updates. An update up is said *deterministic* if for any clock valuation v, there exists at most one valuation v' such that $v' \in up(v)$. It is immediate to check that a local update $up = (up_i)_{1 \leq i \leq k}$ is deterministic if all simple updates up_i are of one of the following form:
  1. $x := c$ with $x \in X$ and $c \in \mathbb{Q}$
  2. $x := y$ with $x, y \in X$
  3. $x := y + c$ with $x, y \in X$ and $c \in \mathbb{Q} \setminus \{0\}$

3 Updatable Timed Automata

We now define the central notion of updatable timed automata. As we explain in details below, these automata extend the classical family of Alur and Dill's timed automata [AD90,AD94].

3.1 The Model

An *updatable timed automaton* over $\mathbb{T}$ is a tuple $\mathcal{A} = (\Sigma, X, Q, T, I, F, B)$, where:

- Σ is a finite alphabet of actions,
- X is a finite set of clocks
- Q is a finite set of states
- $T \subseteq Q \times [\mathcal{C}(X) \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{U}(X)] \times Q$ is a finite set of transitions
- $I \subseteq Q$ is the subset of initial states
- $F \subseteq Q$ is the subset of final states
- $B \subseteq Q$ is the subset of Büchi-repeated states.

The special action ε is called *silent action* and a transition in $Q \times [\mathcal{C}(X) \times \{\varepsilon\} \times \mathcal{U}(X)] \times Q$ is called *silent transition* or *ε-transition*.

If $\mathcal{C} \subseteq \mathcal{C}(X)$ is a subset of clock constraints and $\mathcal{U} \subseteq \mathcal{U}(X)$ a subset of updates, the class ***Uta***$_\varepsilon(\mathcal{C},\mathcal{U})$ denotes the set of all updatable timed automata in which transitions only use clock constraints in $\mathcal{C}$ and updates in $\mathcal{U}$. The subclass of automata which do not use silent transitions is simply written ***Uta***$(\mathcal{C},\mathcal{U})$.

Timed automata, as studied in details by Alur and Dill [AD90,AD94], thus correspond to the classes ***Uta***$_\varepsilon(\mathcal{C}_{df}(X),\mathcal{U}_0(X))$ and ***Uta***$(\mathcal{C}_{df}(X),\mathcal{U}_0(X))$ (where $\mathcal{C}_{df}(X)$ and $\mathcal{U}_0(X)$ are respectively the set of diagonal-free clock constraints and reset updates as defined in section 2).

As for timed automata, a behavior in an updatable timed automaton is obtained through the notion of paths and runs. Let us fix for the rest of this part an updatable timed automaton $\mathcal{A}$. A *path* in $\mathcal{A}$ is a finite or 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 a final state, *or* it is infinite and passes infinitely often through a Büchi-repeated state.
A *run* through the path P from the clock valuation v_0, with $v_0(x) = 0$ for any clock x, is a sequence of the form:

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

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

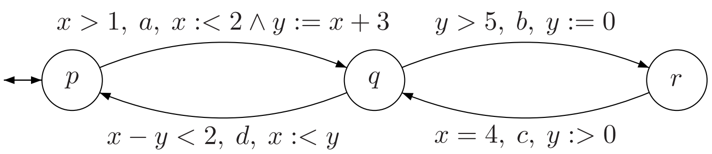
$$\begin{cases} v_{i-1} + (t_i - t_{i-1}) \models \varphi_i \\ v_i \in up_i\left(v_{i-1} + (t_i - t_{i-1})\right) \end{cases}$$

Note that any set $up_i(v_{i-1} + (t_i - t_{i-1}))$ of a run has to be non empty. In the following, to make the notations more compact , we will note such a run

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

The label of such a run is the timed word $w = (a_1, t_1)(a_2, t_2)\ldots$ If the path P is accepting, then this timed word is said to be accepted by $\mathcal{A}$. 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 $L(\mathcal{A})$.

Example 2. Consider the following updatable timed automaton.



A possible (finite) accepting run in this automaton is the following:

$$\langle p, (0,0)\rangle \xrightarrow[1.3]{a} \langle q, (0.2, 4.3)\rangle \xrightarrow[2.1]{b} \langle r, (1,0)\rangle \xrightarrow[5.1]{c} \langle q, (4, 3.1)\rangle \xrightarrow[9.6]{d} \langle p, (7.2, 8.6)\rangle$$

Let us explain this run:

- the transition $\langle p, (0,0)\rangle \xrightarrow[1.3]{a} \langle q, (0.2, 4.3)\rangle$ is possible because after having waited 1.3 units of time, the value of both x and y is 1.3, thus after the update $x :< 2 \wedge y := x + 3$, the valuation $(0.2, 4.3)$ $(4.3 = 1.3 + 3)$ is possible
- the transition $\langle q, (0.2, 4.3)\rangle \xrightarrow[2.1]{b} \langle r, (1,0)\rangle$ is possible because after having waited $2.1 - 1.3 = 0.8$ units of time, the value of x is 1 and the value of y is 0.8, thus after resetting y to 0, we get that the valuation $(1, 0)$ can be reached
- etc...

Remark 1. In [AD94], Alur and Dill claimed that for any timed automaton in $\textit{\textbf{Uta}}_\varepsilon(\mathcal{C}(X), \mathcal{U}_0(X))$ (resp. $\textit{\textbf{Uta}}(\mathcal{C}(X), \mathcal{U}_0(X))$), there exists a timed automaton in $\textit{\textbf{Uta}}_\varepsilon(\mathcal{C}_{df}(X), \mathcal{U}_0(X))$ (resp. $\textit{\textbf{Uta}}(\mathcal{C}_{df}(X), \mathcal{U}_0(X))$) which accepts the same language; the interested reader will find a full proof of this easy fact in [BDGP98].

3.2 Aim of The Paper

The following deep result is the core of the theory of timed automata together with its use for modeling real-time systems. It has been implemented in several tools like CMC [LL98], KRONOS [DOTY96] or UPPAAL [LPY97]. These tools have been intensively used on numerous case studies [DOY94,JLS96,HSLL97,BBP02].

Theorem 1. *[AD90,AD94] The class* $\textit{\textbf{Uta}}_\varepsilon(\mathcal{C}_{df}(X), \mathcal{U}_0(X))$ *is decidable.*

Remind that a class of automata is said *decidable* if there exists an algorithm which, taking as an input an arbitrary automaton of the class, outputs "yes" or "no", depending on whether the language recognized by the automaton is empty or not.

Our goal in this paper is twofold. First, we will study if and how the theorem above can be extended to the class $\textbf{\textit{Uta}}_\varepsilon(\mathcal{C}(X),\mathcal{U}(X))$ and to interesting subclasses. We will then compare the expressive power of these subclasses to the expressive power of automata from $\textbf{\textit{Uta}}_\varepsilon(\mathcal{C}_{df}(X),\mathcal{U}_0(X))$ and $\textbf{\textit{Uta}}(\mathcal{C}_{df}(X),\mathcal{U}_0(X))$.

As it will turn out, it is necessary to distinguish the cases where only diagonal-free clock constraints are used and where arbitrary clock constraints are authorized. Recall that on the contrary, any Alur and Dill's timed automaton using arbitrary clock constraints can be transformed into an other Alur and Dill's timed automaton using only diagonal-free clock constraints (see Remark 1).

4 Undecidability Results

In this section, we first exhibit undecidable classes of updatable timed automata.

Let us first recall briefly that a two counter machine (known sometimes also as a Minsky machine) is a finite set of labeled instructions over two counters c_1 and c_2. There are two types of instructions over counters:

- an *incrementation instruction* of counter $x \in \{c_1, c_2\}$:

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

- a *decrementation (or zero-testing) instruction* of counter $x \in \{c_1, c_2\}$:

$$p:\ \text{if}\ \ x>0\ \begin{cases} \text{then}\ \ x := x-1\ ;\ \ \text{goto}\ \ q \\ \text{else}\quad \text{goto}\ \ r \end{cases} \qquad \text{(where } p,\ q \text{ and } r \text{ are instruction labels)}$$

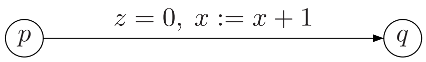
The machine starts at an instruction labeled by s_0 with $c_1 = c_2 = 0$ and stops at a special instruction labeled by HALT. The *halting problem* for a two counter machine consists in deciding whether the machine reaches the instruction HALT.

The following result will be the basis of all our undecidability results on updatable timed automata.

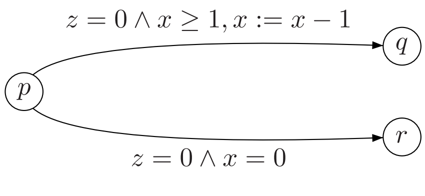
Theorem 2. *[Min67] The halting problem for two counter machines is undecidable.*

Instructions of a two counter machine can easily be simulated by transitions of updatable timed automata. States of the automaton are the labels of the instructions of the two counter machine. The transformation can be done in the following way (the unique action a of the alphabet Σ is not represented):

– **Incrementation of counter x:**



– **Decrementation of counter x:**



where the new clock z ensures that no time can elapse (there is no time progress assumption). Such a clock will be used in all constructions presented in this section. More involved constructions could also be done under the time progress assumption.

Thus, given a two counter machine $\mathcal{M}$, an updatable timed automaton $\mathcal{A}_\mathcal{M} \in \textbf{\textit{Uta}}(\mathcal{C}_{df}(X),\mathcal{U}(X))$ satisfying:

$$\mathcal{M} \text{ halts} \iff L(\mathcal{A}_\mathcal{M}) \neq \varnothing$$

can easily be constructed. We thus obtain:

Proposition 1. *Let X be a set of clocks containing at least 3 clocks. Then, the class* ***Uta***$(\mathcal{C}_{df}(X),\mathcal{U}(X))$ *of updatable timed automata is undecidable.*

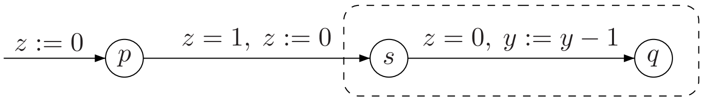
Since any class containing an undecidable subclass is obviously itself undecidable, we get immediately the following corollary:

Corollary 1. *Let X be a set of clocks containing at least 3 clocks. Then, the classes* ***Uta***$(\mathcal{C}(X),\mathcal{U}(X))$*,* ***Uta***$_\varepsilon(\mathcal{C}_{df}(X),\mathcal{U}(X))$ *and* ***Uta***$_\varepsilon(\mathcal{C}(X),\mathcal{U}(X))$ *are undecidable.*

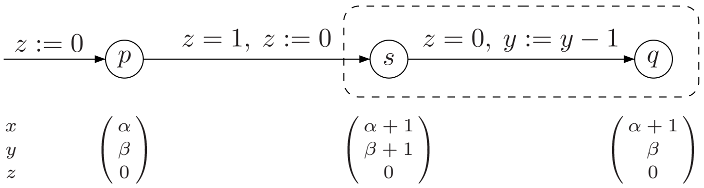
The previous simulations use updates of both types $x := x+1$ and $x := x-1$. We will show that if resets are used, one such type of update is sufficient to build a timed automaton $\mathcal{A}_\mathcal{M}$ as above from a two counter machine $\mathcal{M}$, and thus obtain undecidability results.

Let us first consider updates of the type $x := x-1$, then incrementation of a counter can be simulated as follows:

Incrementation of counter x:

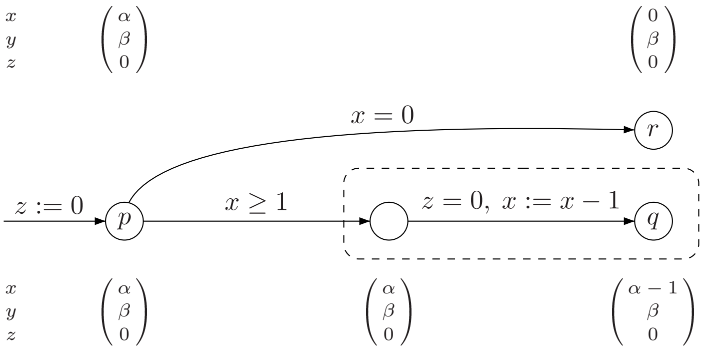


We claim that a run on this path increases the value of clock x of one time unit and keeps unchanged the value of clock y. Indeed, in such a run, the tuple of clock values are of the form (with the order $x,\ y,\ z$ from left to right), $(\alpha,\beta,0)$ when entering state p, $(\alpha+1,\beta+1,0)$ when entering state s and $(\alpha+1,\beta,0)$ when entering state q. In the following, we will represent this by the simple figure below:



The simulation of the decrementation of a counter is identical as the one previously seen. We present it in a quite different and schematic way as follows:

Decrementation of counter x:



If $\mathcal{M}$ is a two counter machine, we can thus construct, as before, a timed automaton $\mathcal{A}_\mathcal{M}$ with only resets to zero and decrementations of clocks and such that

$$\mathcal{M} \text{ halts} \iff L(\mathcal{A}_\mathcal{M}) \neq \varnothing$$

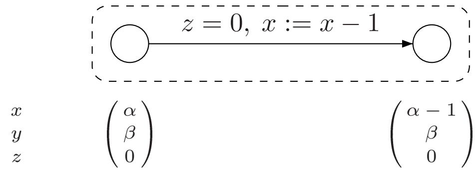
We have thus proven the following result:

Proposition 2. *Let X be a set of clocks containing at least 3 clocks. Let $\mathcal{U}$ be a set of updates containing both $\mathcal{U}_0(X)$ and $\{x := x-1 \mid x \in X\}$. Then the class* **Uta**$(\mathcal{C}_{df}(X), \mathcal{U})$ *is undecidable.*

Remark 2. Note that the previous result can be strengthened because in the construction all reset operations are performed when the clock we want to reset is 0 or 1, they can thus be replaced by decrementations.

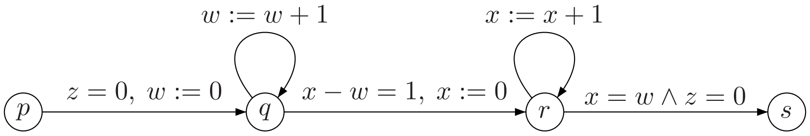
Up to now, all the timed automata constructed for undecidability proofs only have diagonal-free clock constraints (*i.e.* constraints in $\mathcal{C}_{df}(X)$). In the remainder of this section, some of the constructions we will make for proving some undecidability results will also use diagonal clock constraints (not in $\mathcal{C}_{df}(X)$ but in $\mathcal{C}(X)$), and as a byproduct of the results in section 5, it will appear that in these cases, the classes obtained by replacing $\mathcal{C}(X)$ by $\mathcal{C}_{df}(X)$ are indeed decidable.

From the constructions above, we can notice that it is no more necessary to simulate a whole two counter machine in order to prove undecidability results, but that, if resets are allowed, it is sufficient to be able to simulate executions of the form:

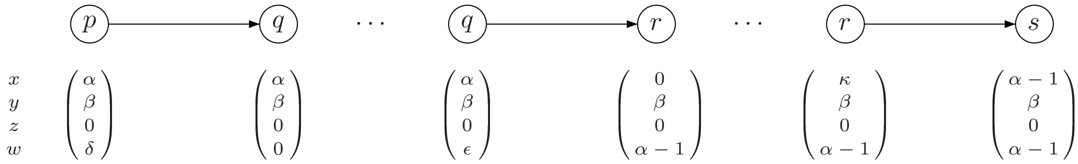


$(\star)$

We first claim that such an execution can be simulated using only updates from the set $\mathcal{U}_0(X) \cup \{x := x+1 \mid x \in X\}$. Indeed, consider the (part of) timed automaton below:



The sequence of clock valuations for a run along this path can be described by:



Such a run thus simulates an execution through a transition $(\star)$.

Proposition 3. *Let X be a set of clocks containing at least 4 clocks. Let $\mathcal{U}$ be a set of updates containing both $\mathcal{U}_0(X)$ and $\{x := x+1 \mid x \in X\}$. Then the class* **Uta**$(\mathcal{C}(X), \mathcal{U})$ *is undecidable.*

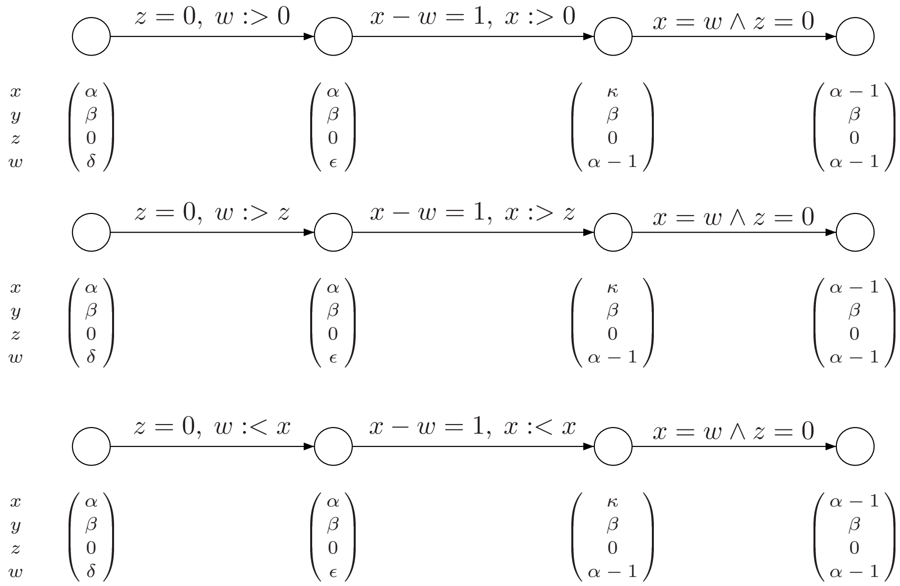
The next undecidability results are obtained thanks to very similar techniques.

Proposition 4. *Let X be a set of clocks containing at least 4 clocks. Let $\mathcal{U}$ be a set of updates containing both $\mathcal{U}_0(X)$ and either*

- $\{x :> 0 \mid x \in X\}$ *or*
- $\{x :> y \mid x, y \in X\}$ *or*
- $\{x :< y \mid x, y \in X\}$.

Then the class **Uta**$(\mathcal{C}(X), \mathcal{U})$ *is undecidable.*

Proof. As before, we simulate the execution through a transition ($\star$) using parts of timed automata. The three automata below correspond respectively to the three sets of updates of the proposition:



Hence, we get the undecidability results announced in the proposition. ☐

From the above results we can prove some more undecidability results. We summarize all the results in Table 1.

	$\mathcal{U}_0(X) \cup \ldots$	**Diagonal-free constraints**	**General constraints**
1	$x := c,\ x := y$	?	?
2	$x := x + 1$		Undecidable
3	$x := y + c$		
4	$x := x - 1$	Undecidable	
5	$x :< c$	?	?
6	$x :> c$		Undecidable
7	$x :\sim y + c$		
8	$y + c <: x :< y + d$		
9	$y + c <: x :< z + d$	Undecidable	

with $\sim \in \{\leq, <, >, \geq\}$ and $c,\ d \in \mathbb{Q}^+$

Table 1. Undecidability results

Lines 2 and 4 correspond exactly to propositions 3 and 2 respectively. Line 3 is just an extension of Line 2. The second column of lines 6, 7, 8 and 9 are direct consequences from proposition 4. The remaining case is the one where we allow diagonal-free clock constraints and updates of the form $y + c <: x :< z + d$, as described on line 9. The corresponding model which also allows in addition diagonal clock constraints is undecidable (see above), we just need to be able to replace diagonal clock constraints by updates of the form $y + c <: x :< z + d$. Assume there is a clock constraint $x - y < c$, its truth or falsity is equivalent to the existence of a value α taken in the real interval $]x; y + c[$. Adding a new clock z, it becomes equivalent to having an update $x <: z :< y + c$.

The next section is devoted to the study of classes marked with "?" and we will see that the emptiness problem is in fact decidable for these remaining classes.

5 Decidability Results

In this section, we extend the decidability result of Theorem 1 to other subclasses of updatable timed automata. Recall that the principle of this deep result relies on the construction, for any timed automaton $\mathcal{A}$, of a finite untimed automaton $\mathcal{B}$ accepting exactly the language $\text{UNTIME}(L(\mathcal{A}))$ where

$$\text{UNTIME}(L(\mathcal{A})) = \{\sigma \in \Sigma^\infty \mid \text{there exists a time sequence } \tau \text{ s.t. } (\sigma, \tau) \in L(\mathcal{A})\}$$

The emptiness of $L(\mathcal{A})$ is obviously equivalent to the emptiness of $\text{UNTIME}(L(\mathcal{A}))$, so the result follows from the decidability of the emptiness checking problem for untimed finite automata (see e.g. [HU79]).

We will generalize the construction of Theorem 1. Let us first define the notion of regions and region graphs.

5.1 Regions and Region Automaton

Let X be a finite set of clocks. We consider a finite partitioning $\mathcal{R}$ of $\mathbb{T}^X$. For each valuation $v \in \mathbb{T}^X$, the unique element of $\mathcal{R}$ that contains v is denoted by $[v]_\mathcal{R}$. We define the successors of R, $\texttt{Succ}(R) \subseteq \mathcal{R}$, in the following natural way:

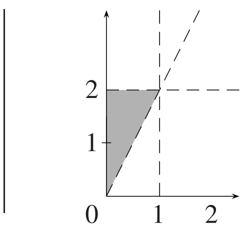
$$R' \in \texttt{Succ}(R) \text{ if } \exists v \in R,\ \exists t \in \mathbb{T} \text{ s.t. } [v+t]_\mathcal{R} = R'$$

We say that such a finite partition is a *set of regions* whenever the following condition holds:

$$R' \in \texttt{Succ}(R) \iff \forall v \in R,\ \exists t \in \mathbb{T} \text{ s.t. } [v+t]_\mathcal{R} = R' \tag{$\star\star$}$$

This natural condition assesses that the equivalence relation defined by the $\mathcal{R}$ partitioning is stable with time elapsing. Roughly, this means that two equivalent valuations stay equivalent while time is elapsing. Let us note that this condition is not satisfied by any finite partition of $\mathbb{T}^X$ as illustrated by the following counter-example.

Example 3. Let us consider the partition of $\mathbb{T}^2$ drawn on the figure beside. Condition ($\star\star$) is not satisfied by the gray region. Indeed, from valuation $(0,5;1,8)$, when time elapses it is possible to reach the valuation $(0,7;2)$ and thus the region defined by the constraints $0 < x < 1 \wedge y = 2$. But this region can not be reached from valuation $(0,5;1,1)$.



Let $\mathcal{U} \subseteq \mathcal{U}(X)$ 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 any region R onto the set $\{R' \in \mathcal{R} \mid up(R) \cap R' \neq \varnothing\}$. The set of regions $\mathcal{R}$ is said *compatible* with $\mathcal{U}$ if whenever a valuation $\overline{v}' \in R'$ is reachable from a valuation $\overline{v} \in R$ by some $\widehat{up}$ then R' is reachable from any $v \in R$ by the same $\widehat{up}$. Formally, we require:

$$R' \in \widehat{up}(R) \implies \forall v \in R,\ \exists v' \in R' \text{ s.t. } v' \in up(v) \tag{$\star\star\star$}$$

Note that this condition has an interpretation similar to the one done for condition ($\star\star$). Of course these conditions are related to some kind of bisimulation property, see the remark below.

Remark 3. If the transition relations $(\hookrightarrow_{up})_{up}$ on $\mathbb{T}^X$ are defined by

$$v \hookrightarrow_{up} v' \iff v' \in up(v)$$

and the relation $\rho_\mathcal{R}$ by

$$v\ \rho_\mathcal{R}\ v' \iff [v]_\mathcal{R} = [v']_\mathcal{R}$$

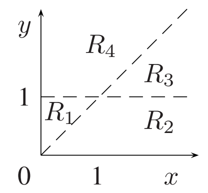
then the condition ($\star\star\star$) assesses that $\rho_\mathcal{R}$ is a bisimulation with respect to the relations $(\hookrightarrow_{up})_{up}$.

Whenever a set of regions $\mathcal{R}$ is compatible with a set of updates $\mathcal{U}$, we define the *region graph* associated with $\mathcal{R}$ and $\mathcal{U}$ as the graph whose set of nodes is $\mathcal{R}$ and whose edges are of two distinct types:

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

Example 4. Let us consider the set of four regions $\mathcal{R}$ defined by the following equations:

$$\begin{array}{cccc} R_1 & R_2 & R_3 & R_4 \\ \begin{pmatrix} 0 \le x < 1 \\ 0 \le y \le 1 \\ x < y \end{pmatrix} & \begin{pmatrix} x \ge 0 \\ 0 \le y \le 1 \\ x \ge y \end{pmatrix} & \begin{pmatrix} x > 1 \\ y > 1 \\ x \ge y \end{pmatrix} & \begin{pmatrix} x \ge 0 \\ y > 1 \\ x < y \end{pmatrix} \end{array}$$



It is easy to verify that $\mathcal{R}$ is compatible with the set of updates $\mathcal{U} = \{x := 1, y := 0\}$. The region graph associated with $\mathcal{R}$ and $\mathcal{U}$ is represented below on Figure 1.

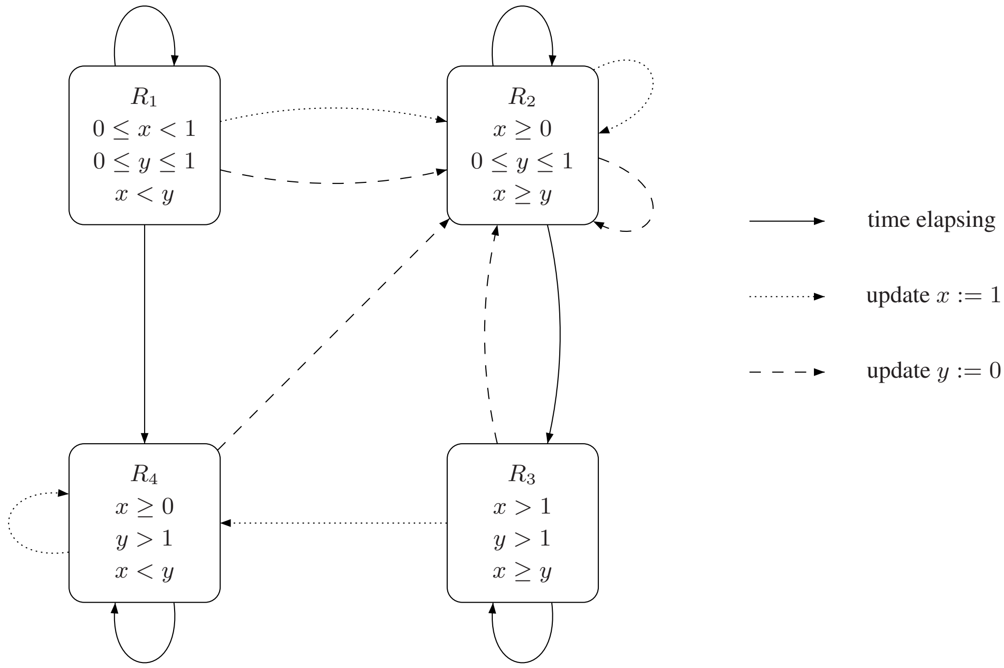


Fig. 1. A simple example of region graph

Finally, let $\mathcal{C} \subseteq \mathcal{C}(X)$ be a finite set of clock constraints. A set of regions $\mathcal{R}$ is said to be *compatible* with $\mathcal{C}$ if for every clock constraint $\varphi \in \mathcal{C}$ and for every region R, either $R \subseteq \varphi$ or $R \subseteq \neg\varphi$.

Let now $\mathcal{A} = (\Sigma, X, Q, T, I, F, B)$ be a timed automaton in some class $\boldsymbol{Uta}(\mathcal{C}, \mathcal{U})$ and let $\mathcal{R}$ be a family of regions compatible with $\mathcal{C}$ and $\mathcal{U}$. We define the *region automaton* $\Gamma_{\mathcal{R}}(\mathcal{A})$ associated with $\mathcal{A}$ and $\mathcal{R}$, as the following finite (untimed) automaton:

– Its set of locations is $Q \times \mathcal{R}$.
  - The initial locations are $(q_0, \mathbf{0})$ where $q_0 \in I$ is initial and $\mathbf{0}$ is the unique region containing the valuation where all clocks are set to zero
  - The final locations are (f, R) where f is final in $\mathcal{A}$ and R is any region
  - The repeated locations are (r, R) where r is repeated in $\mathcal{A}$ and R is any region
– Its transitions are defined by $(q, R) \xrightarrow{a} (q', R')$ if there exists a region $\widehat{R}$ and a transition $q \xrightarrow{\varphi,a,up} q'$ in $\mathcal{A}$ such that:
  - $R \longrightarrow \widehat{R}$ is a transition of the region graph,
  - $\widehat{R} \subseteq \varphi$
  - $\widehat{R} \longrightarrow_{up} R'$ is a transition of the region graph.

Under conditions $(\star\star)$ and $(\star\star\star)$, the region automaton is an interesting abstraction of the original automaton in the sense that we obtain a result similar to the one of Theorem 1.

Proposition 5. *Let $\mathcal{A}$ be a timed automaton in **Uta**$(\mathcal{C},\mathcal{U})$ where $\mathcal{C}$ (resp. $\mathcal{U}$) is a finite set of clock constraints (resp. of updates). Let $\mathcal{R}$ be a set of regions compatible with $\mathcal{C}$ and $\mathcal{U}$. Then the finite automaton $\Gamma_{\mathcal{R}}(\mathcal{A})$ accepts the language* $\textsc{Untime}(L(\mathcal{A}))$.

Proof. Assume that $\mathcal{A} = (\Sigma, Q, T, I, F, R, X)$.
Let us take a run in $\mathcal{A}$

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

For $i \geq 0$, let us define $R_i = [v_i]_{\mathcal{R}}$ and $\widehat{R_i} = [v_i + t_{i+1} - t_i]_{\mathcal{R}}$. It holds that $\widehat{R_i} \in \mathtt{Succ}(R_i)$ and, since $v_{i+1} \in up_{i+1}(v_i + t_{i+1})$, $R_{i+1} \in \widehat{up}_i(\widehat{R_i})$. Moreover, $v_i + t_{i+1} \models \varphi_{i+1}$ and since $\mathcal{R}$ is compatible with $\mathcal{C}$, we deduce that $\widehat{R_i} \subseteq \varphi_{i+1}$. Therefore, from the definition,

$$\langle q_0, R_0\rangle \xrightarrow{a_1} \langle q_1, R_1\rangle \xrightarrow{a_2} \cdots$$

is an accepting path of $\Gamma_{\mathcal{R}}(\mathcal{A})$. Hence $\textsc{Untime}(L(\mathcal{A})) \subseteq L(\Gamma_{\mathcal{R}}(\mathcal{A}))$ holds.
Conversely, let us consider a run in $\Gamma_{\mathcal{R}}(\mathcal{A})$,

$$\langle q_0, R_0\rangle \xrightarrow{a_1} \langle q_1, R_1\rangle \xrightarrow{a_2} \cdots$$

We set $v_0 = 0$ and assume that we have already constructed sequences $(v_i)_{0\leq i<n}$ and $(t_i)_{1\leq i<n}$ such that $v_i \in R_i$ and such that the following is a run of $\mathcal{A}$

$$\langle q_0, v_0\rangle \xrightarrow[t_1]{\varphi_1,a_1,up_1} \langle q_1, v_1\rangle \cdots \xrightarrow[t_{i-1}]{\varphi_{i-1},a_{i-1},up_{i-1}} \langle q_{i-1}, v_{i-1}\rangle$$

Since $\langle q_{i-1}, R_{i-1}\rangle \xrightarrow{a_i} \langle q_i, R_i\rangle$ is a transition of $\Gamma_{\mathcal{R}}(\mathcal{A})$, there exists by definition a region $\widehat{R}$ and a transition $(q_{i-1}, \varphi_i, a_i, up_i, q_i)$ in $\mathcal{A}$ such that

– $R_{i-1} \longrightarrow \widehat{R}$ is a transition of the region graph,
– $\widehat{R} \subseteq \varphi_i$
– $\widehat{R} \longrightarrow_{up_i} R_i$ is a transition of the region graph.

From $v_{i-1} \in R_{i-1}$ and the fact that the set of regions $\mathcal{R}$ satisfies $(\star\star)$, it follows that there exists some $t_i \in \mathbb{T}$ such that $v_{i-1} + t_i - t_{i-1} \in \widehat{R}$. Now, from the hypothesis that $\mathcal{R}$ is compatible with up_i, we deduce that there exists some valuation v_i such that $v_i \in up_i(v_{i-1} + t_i - t_{i-1})$. Hence the following is a path in $\mathcal{A}$

$$\langle q_0, v_0\rangle \xrightarrow[t_1]{\varphi_1,a_1,up_1} \langle q_1, v_1\rangle \cdots \xrightarrow[t_{i-1}]{\varphi_{i-1},a_{i-1},up_{i-1}} \langle q_{i-1}, v_{i-1}\rangle \xrightarrow{\varphi_i,a_i,up_i} \langle q_i, v_i\rangle$$

Therefore, we construct by induction a path in $\mathcal{A}$,

$$\langle q_0, v_0\rangle \xrightarrow[t_1]{\varphi_1,a_1,up_1} \langle q_1, v_1\rangle \cdots \langle q_{i-1}, v_{i-1}\rangle \xrightarrow[t_i]{\varphi_i,a_i,up_i} \langle q_i, v_i\rangle \cdots$$

We thus have $L(\Gamma_{\mathcal{R}}(\mathcal{A})) \subseteq \textsc{Untime}(L(\mathcal{A}))$ which concludes the proof of this proposition. $\square$

Since the emptiness checking problem for untimed (Büchi or with a finite acceptance condition) automaton is decidable (see e.g. [HU79]), the previous proposition leads to the next theorem.

Theorem 3. *Let $\mathcal{C}$ (resp. $\mathcal{U}$) be a finite set of clock constraints (resp. of updates). Assume there exists a set of regions $\mathcal{R}$ such that $\mathcal{R}$ is compatible with $\mathcal{C}$ and $\mathcal{U}$, then the class* **Uta**$(\mathcal{C},\mathcal{U})$ *is decidable.*

This theorem is of course fundamental, but it does not exhibit any real decidable class of updatable automata for which we can decide emptiness. Indeed, we need to construct sets of clock constraints $\mathcal{C}$ and sets of updates $\mathcal{U}$, together with sets of regions $\mathcal{R}$ such that $\mathcal{R}$ is compatible with both $\mathcal{C}$ and $\mathcal{U}$.

As mentioned before, we quickly had the intuition that diagonal-free and general clock constraints do not lead to the same (un)decidability properties. This is the reason why we proceed by distinguishing classes of updatable timed automata according to the type of constraints, diagonal-free or not.

First we need a lemma claiming that we can restrict our investigations to updatable timed automata which use integer (and not rational) constants only. The result is a trivial extension of a remark proposed and proven by Alur and Dill for classical timed automata (*cf* lemma 4.1, page 15 of [AD94]).

Lemma 1. *Let $\mathcal{A}$ be a timed automaton and let λ be a positive rational constant. Let $\lambda\mathcal{A}$ be the timed automaton obtained by replacing all the constants μ of the clock constraints or the updates of $\mathcal{A}$ by the product $\lambda\mu$. Then the language $L(\lambda\mathcal{A})$ equals $\lambda L(\mathcal{A})$ where $\lambda L(\mathcal{A}) = \{(a_i, \lambda t_i)_{i\geq 0} \mid (a_i, t_i)_{i\geq 0} \in L(\mathcal{A})\}$.*

Hence, given a timed automaton $\mathcal{A}$ and a constant $\lambda \in \mathbb{Q}^+$, the emptiness of $L(\mathcal{A})$ is equivalent to the one of $L(\lambda\mathcal{A})$. But if we consider the *lcm* m of all the constants used by $\mathcal{A}$, the automaton $m\mathcal{A}$ deals only with integer constants. Hence, when considering emptiness, we can assume without loss of generality that all the constants appearing in (updatable) timed automata are integers. We will do such an assumption for the rest of this section.

5.2 Decidable Classes of Diagonal-Free Updatable Timed Automata

In this section, we consider diagonal-free clock constraints only, on a set of clocks X. We first construct a set of regions suitable for these constraints. For each clock $x \in X$, we consider an integer constant c_x and we define the set of intervals:

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

Now 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[6] on $X_0 = \{x \in X \mid I_x$ is an interval of the form $]c; c+1[\}$

The region associated with α is defined as the following set of valuations:

$$\left\{ v \in \mathbb{T}^X \;\middle|\; \begin{array}{l} \forall x \in X,\ v(x) \in I_x \text{ and} \\ \forall x, y \in X_0,\ x \prec y \iff \mathsf{frac}(v(x)) \leq \mathsf{frac}(v(y)) \end{array} \right\}$$

In the sequel, we will refer to this set as "**the region** α".

Remark 4. The finite set $\mathcal{R}_{(c_x)_{x\in X}}$ of all such regions forms a partition of $\mathbb{T}^X$. Note that it is exactly (with slightly distinct notations) the set of regions used by Alur and Dill in their seminal paper [AD94]. Hence the following lemma, which claims that this set verifies the condition $(\star\star)$, is not an original result and we prove it here only for the sake of completeness.

[6] Recall that a preorder is a reflexive and transitive relation. If in addition this preorder is antisymmetric, it is an order.

Lemma 2. *The set $\mathcal{R}_{(c_x)_{x \in X}}$ is a set of regions.*

Proof. Assume that $\alpha = ((I_x)_{x \in X}, \prec)$. If for all x, $I_x =]c_x; +\infty[$, then obviously

$$\forall v \in \alpha, \forall t \in \mathbb{T}, v + t \in \alpha$$

and thus $\texttt{Succ}(\alpha) = \{\alpha\}$. Otherwise, there exists at least a region $\alpha' \neq \alpha$ such that $\alpha' \in \texttt{Succ}(\alpha)$. Among these regions we define the "closest" region to α, *i.e.* the region $\alpha_{\texttt{Succ}}$ such that

- $\alpha_{\texttt{Succ}} \in \texttt{Succ}(\alpha)$, and
- $\forall v \in \alpha, \forall t \in \mathbb{T}$, if $v + t \notin \alpha$ then $\exists t' \leq t$ such that $v + t' \in \alpha_{\texttt{Succ}}$.

The region $\alpha_{\texttt{Succ}} = ((I'_x)_{x \in X}, \prec')$ can be characterized as follows. Let $Z = \{x \in X \mid I_x \text{ is of the form } [c]\}$. We distinguish two cases:

1. If $Z \neq \varnothing$, then
   - $I'_x = \begin{cases} I_x & \text{if } x \notin Z \\ ]c; c+1[& \text{if } x \in Z \text{ and } I_x = [c] \text{ with } 0 \leq c < c_x \\ ]c_x; +\infty[& \text{if } x \in Z \text{ and } I_x = [c_x] \end{cases}$
   - $x \prec' y$ if either $x \prec y$ or $I_x = [c]$ with $0 \leq c < c_x$ and I'_y is of the form $]d; d+1[$

2. If $Z = \varnothing$, let M be the set of maximal elements of $\prec$, *i.e.*

$$M = \{x \in X_0 \mid \forall z \in X_0,\ x \prec z \implies z \prec x\}$$

   Then,
   - $I'_x = \begin{cases} I_x & \text{if } y \notin M \\ [c+1] & \text{if } x \in M \text{ and } I_x =]c; c+1[\text{ with } 0 \leq c < c_x \end{cases}$
   - $\prec'$ is the restriction of $\prec$ to $\{x \in X \mid I'_x \text{ is of the form }]d; d+1[\}$

We claim now that

$$\forall v \in \alpha,\ \exists t \in \mathbb{T} \text{ such that } v + t \in \alpha_{succ}$$

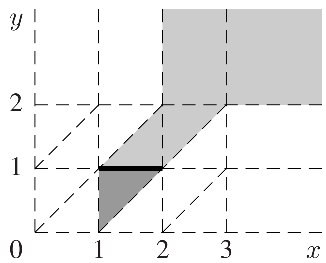
Indeed, let v be a valuation in α,

1. If $Z \neq \varnothing$, then let $\tau = \min\left(\{1 - \mathsf{frac}(v(x)) \mid I_x \text{ is of the form }]c; c+1[\}\right)$. Then the valuation $v + \frac{1}{2}\tau$ is in the region α_{succ}.
2. If $Z = \varnothing$, then let $\tau = 1 - \mathsf{frac}(v(x))$ for any $x \in M$. Then the valuation $v + \tau$ is in α_{succ}.

Now, we get by an immediate induction that the set $\mathcal{R}_{(c_x)_{x \in X}}$ verifies condition ($\star\star$) which achieves the proof of the lemma. □

Example 5. 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 dark gray region is defined by $I_x =]1; 2[$, $I_y =]0; 1[$, and $x \prec y$ and $y \not\prec x$.

The immediate successor region of this (dark) gray region is defined by $I_x =]1; 2[$ and $I_y = [1]$ (drawn as a thick line). The other successor regions are drawn in light gray.



The sets of regions we consider is now defined, the following result about their compatibility with sets of diagonal-free clock constraints is immediate.

Proposition 6. *Let $\mathcal{C} \subseteq \mathcal{C}_{df}(X)$ be such that for any clock constraint $x \sim c$ of $\mathcal{C}$, it holds that $c \leq c_x$. Then the set of regions $\mathcal{R}_{(c_x)_{x \in X}}$ is compatible with $\mathcal{C}$.*

Note that the result does not hold anymore for an arbitrary set of constraints included in $\mathcal{C}(X)$. For instance, in the example above, the region $((]3;+\infty[,]2;+\infty[),\varnothing)$ is neither included in $x-y \leq 1$ nor in $x-y \geq 1$.

We now investigate the compatibility of $\mathcal{R}_{(c_x)_{x \in X}}$ and sets of updates $\mathcal{U}$. We first consider the case of **simple** updates. Recall that a simple update (*cf* section 2.3) is an update of the form $z :\sim c$ or $z :\sim y + c$ where y and z are clocks, $\sim\ \in \{<, \leq, =, \geq, >\}$ and c is an (integer) constant. Note that even if the set $\mathcal{R}_{(c_x)_{x \in X}}$ is the one used by Alur and Dill (*cf* Remark 4), its compatibility with all the updates distinct from resets (*i.e.* of the form $x := 0$) is not proven yet.

Lemma 3. *Let $\mathcal{R}_{(c_x)_{x \in X}}$ be a set of regions. This set of regions is compatible with any simple update $z :\sim c$ such that $c \leq c_z$ and with any simple update $z :\sim y+c$ such that $c_z \leq c_y+c$, with $\sim \in \{=, \neq, <, >, \leq, \geq\}$.*

Proof. Assume that $\alpha = ((I_x)_{x \in X}, \prec)$ is a region of $\mathcal{R}_{(c_x)_{x \in X}}$. Recall that $\prec$ is thus a total preorder on $X_0 = \{x \in X \mid I_x \text{ is an interval of the form }]c;c+1[\}$. Let up be a simple update over z. We first characterize the regions of $\widehat{up}(\alpha)$.

Let $\alpha' = ((I'_x)_{x \in X}, \prec')$ (where $\prec'$ is a total preorder on X'_0). Then α' is in $\widehat{up}(\alpha)$ if $I'_x = I_x$ for all $x \neq z$ and:

if *up* is *z:* $\sim$ *c*: I'_z can be any interval of $\mathcal{I}_z$ which intersects $\{\gamma \in \mathbb{T} \mid \gamma \sim c\}$ and

- either I'_z is of the form $[d]$ or $]c_z;+\infty[$ and thus
  - $X'_0 = X_0 \setminus \{z\}$
  - $\prec' = \prec \cap (X'_0 \times X'_0)$
- either I'_z is of the form $]d;d+1[$ and thus
  - $X'_0 = X_0 \cup \{z\}$
  - $\prec'$ is any total preorder on X'_0 which coincides with $\prec$ on $X'_0 \setminus \{z\}$.

if *up* is *z:* $\sim$ *y+c* with *c* $\in \mathbb{Z}$: I'_z can be any interval of $\mathcal{I}_z$ such that there exists $a \in I'_z$, $b \in I_y$ with $a \sim b + c$ and

- either I'_z is of the form $[d]$ or $]c_z;+\infty[$
  - $X'_0 = X_0 \setminus \{z\}$
  - $\prec' = \prec \cap (X'_0 \times X'_0)$
- either I'_z is of the form $]d;d+1[$,
  - $X'_0 = X_0 \cup \{z\}$
    - ∗ 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 we have to take care of the relative values of $\mathsf{frac}(v'(y))$ and $\mathsf{frac}(v'(z))$ when $(I_y + c) \cap I'_z \neq \varnothing$:
      - · either $(I_y + c) \cap I'_z = \varnothing$ and $\prec'$ is any total preorder on X'_0 which coincides with $\prec$ on $X_0 \setminus \{z\}$
      - · either $(I_y + c) \cap I'_z \neq \varnothing$
        Note that from the inequality $c_z \leq c_y + c$, this condition implies that $I_y + c \subseteq I'_z$. In that case, $\prec'$ is any total preorder on X'_0 which coincides with $\prec$ on $X'_0 \setminus \{z\}$ and verifies:

$\cdot\ z \prec' y$ and $y \prec' z$ if $\sim$ is $=$
$\cdot\ z \prec' y$ and $y \not\prec' z$ if $\sim$ is $<$
$\cdot\ z \prec' y$ if $\sim$ is $\leq$
$\cdot\ y \prec' z$ if $\sim$ is $\geq$
$\cdot\ z \not\prec' y$ and $y \prec' z$ if $\sim$ is $>$
$\cdot$ ($z \prec' y$ and $y \not\prec' z$) or ($z \not\prec' y$ and $y \prec' z$) if $\sim$ is $\neq$

From this construction, it is now easy to check that condition $(\star\star\star)$ holds *i.e.* that for any $v \in \alpha$ and any $\alpha' \in \widehat{up}(\alpha)$, there exists $v' \in \alpha' \cap up(v)$. Indeed, since up is a local update over z, $v'(x) = v(x)$ for all $x \neq z$ and we just have to define $v'(z)$.

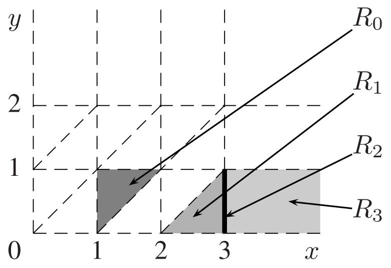
1. If $z \notin X'_0$, then
   (a) If $I'_z = [c]$, $v'(z)$ is of course set to c.
   (b) If $I'_z =]c_z; \infty[$, since $I_y + c \subseteq I'_z$, $v(y) + c$ belongs to the open interval $]c_z; \infty[$. Hence, whatever $\sim$ in $\{=, \neq, <, \leq, >, \geq\}$, there exists some value α such that $\alpha \sim v(y) + c$.We thus set $v'(z)\alpha$
2. if $z \in X'_0$, then
   (a) If $x \prec' z$ and $z \prec' x$ for some x, then $v'(z) = d + \mathsf{frac}(v'(x))$ with $I'_z =]d; d+1[$
   (b) If, for any clock x, either $x \not\prec' z$ or $z \not\prec' x$, then $v'(z) = d + \tau$ with $I'_z =]d; d+1[$ and

$$\max\{\mathsf{frac}(v'(x)) \mid x \prec' z\} < \tau < \min\{\mathsf{frac}(v'(x)) \mid z \prec' x\}$$

Note that since the time domain is assumed to be dense, there always exists (an infinity of) such τ.

In all cases, it holds $v' \in \alpha' \cap up(\alpha)$ and the lemma is proven. $\square$

Example 6. Let us consider the case where $X = \{x, y\}$, and the constants c_x and c_y are given by $c_x = 3$ and $c_y = 2$. The set of regions $\mathcal{R}_{c_x,c_y}$ is represented on the figure beside. The image of the region R_0, $I_x =]1, 2[$, $I_y =]0, 1[$, $x \prec y$ by the update $x :> y + 2$ is composed of three regions, namely:



- Region R_1: $I'_x =]2; 3[$, $I'_y =]0; 1[$ and $y \prec' x$
- Region R_2: $I'_x = [3]$, $I'_y =]0; 1[$
- Region R_3: $I'_x =]3; +\infty[$, $I'_y =]0; 1[$

Consider now a local update $up = (up_i)_{1 \leq i \leq k}$ where each up_i is a simple update over some clock x_i. Let also $\mathcal{R}_{(c_x)_{x \in X}}$ be a set of regions as defined above. It could happen that each up_i is compatible with this set of regions whereas up itself is not compatible any more. Indeed, let us define $X = \{x, y, z\}$, $c_x = 2$, $c_y = c_z = 1$, $\alpha((]2; \infty[,]1; \infty[, \{1\}), \varnothing)$ and $\alpha'((]2; \infty[,]1; \infty[,]1; \infty[), \varnothing)$. Finally, let up_1 be the update $z :< x$ and up_2 be the update $z :> y$. It is obvious that

$$\forall v' \in \alpha', \exists v_1, v_2 \in \alpha \text{ s.t. } v_1 \in up_1(v) \text{ and } v_2 \in up_2(v)$$

However the two valuations $(2.3, 1.1, 1)$ and $(2.3, 3.4, 1)$ both belong to α and $(2.3, 1.1, 1.8)$ is in $\alpha' \cap up((2.3, 1.1, 1))$ whereas $up((2.3, 3.4, 1)) = \varnothing$.

Therefore, in order to get local updates compatible with the sets of regions of the form $\mathcal{R}_{(c_x)_{x \in X}}$, we need to restrict the local updates we consider. From the counterexample just above, it appears that a given clock can not be set to an interval in which the lower and upper bounds depend on two distinct clocks. Moreover, from lemma 3, we need to restrict the constants that are used by the simple updates. This naturally leads to the following definition:

Definition 1. *Let $(c_x)_{x\in X}$ be integer constants. The set $\mathcal{U}_{(c_x)_{x\in X}}$ is constituted of updates of the form $up = \bigwedge_{x\in X} up_x$ where, for each clock $x \in X$, up_x is a local update over the clock x defined by one of the four following abstract grammars:*

- det_x ::= $x := c \mid x := z + d$
  with $z \in X$, $c, d \in \mathbb{Z}$, $c \leq c_x$ and $c_x \leq c_z + d$
- inf_x ::= $x :\lhd c \mid x :< z + d \mid inf_x \wedge inf_x$
  with $\lhd \in \{<, \leq\}$, $z \in X$, $c, d \in \mathbb{Z}$, $c \leq c_x$ and $c_x \leq c_z + d$
- sup_x ::= $x :\rhd c \mid x :> z + d \mid sup_x \wedge sup_x$
  with $\rhd \in \{>, \geq\}$, $z \in X$, $c, d \in \mathbb{Z}$, $c \leq c_x$ and $c_x \leq c_z + d$
- 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, c, c', d, d' are in $\mathbb{Z}$,
  $c, c' \leq c_x$, $c_x \leq c_z + d'$ and $c_x \leq c_z + c'$

The basis of an update $up = \bigwedge_{x\in X} up_x$ of $\mathcal{U}_{(c_x)_{x\in X}}$ is intuitively the set Y of clocks which can be modified by the update up. Formally, this set Y is defined through its complement:

$$X \setminus Y = \{z \in X \mid up_z \text{ is equal to } z := z\}$$

The first step for proving the compatibility of $\mathcal{R}_{(c_x)_{x\in X}}$ and $\mathcal{U}_{(c_x)_{x\in X}}$ is given by the following lemma. Its proof is very similar to the one of lemma 3 and therefore left to the reader.

Lemma 4. *Let $\mathcal{R}_{(c_x)_{x\in X}}$ be a set of regions. This set of regions is compatible with any local update of $\mathcal{U}_{(c_x)_{x\in X}}$ which basis is reduced to a single clock $\{x\}$.*

We can now state our main result concerning the compatibility of sets of regions and sets of updates, in the case of diagonal-free updatable timed automata.

Proposition 7. *Let $(c_x)_{x\in X}$ be integer constants. Then the set of regions $\mathcal{R}_{(c_x)_{x\in X}}$ is compatible with the set of updates $\mathcal{U}_{(c_x)_{x\in X}}$.*

Proof. Let $\alpha = ((I_y)_{y\in X}, \prec), \alpha' = ((I'_y)_{y\in X}, \prec')$ be two regions of $\mathcal{R}_{(c_x)_{x\in X}}$ and up be an update of $\mathcal{U}_{(c_x)_{x\in X}}$ such that $\alpha' \in \widehat{up}(\alpha)$ *i.e.* there exists some valuations $v \in \alpha$ and $v' \in \alpha'$ such that $v' \in up(v)$. For any clock x, let v_x be the valuation defined by:

$$v_x(y) = \begin{cases} v(y) & \text{if } y \neq x \\ v'(x) & \text{if } y = x \end{cases}$$

and let $\alpha_x = ((I_y^{(x)})_{y\in X}, \prec^{(x)})$ be the (unique) region of $\mathcal{R}_{(c_x)_{x\in X}}$ containing v_x.

Now let w be a valuation in α. From lemma 4, $\mathcal{R}_{(c_x)_{x\in X}}$ is compatible with up_x, thus, for any clock x, there exists some valuation $w_x \in up_x(w) \cap \alpha_x$. We now define the valuation w' by setting

$$w'(y) = w_y(y) \text{ for any clock } y$$

From the definition of a local update, it turns out that $w' \in up(w)$. We claim that $w' \in \alpha'$, too. Indeed, for any clock y, $w'(y) = w_y(y) \in I_y^{(y)} = I'_y$. It remains to show that the sequence $\mathsf{frac}(w'(x))_{x\in X}$ verifies the conditions given by the preorder $\prec'$. To this purpose, it is sufficient to prove that the preorder $\prec'$ (which is given, a priori, by the valuation v') can be defined from $\prec$ and the sequence $(\prec^{(x)})_{x\in X}$.

From the constructions given in lemma 3, which can be extended to prove lemma 4, it is easy to check that the preorder $\prec'$ can be computed as follows.

Let X' be a disjoint copy of the set of clocks X. We first define a sequence $(\overline{\prec}^{(x)})_{x\in X}$ of preorders on the set $X \cup X'$. Intuitively $\overline{\prec}^{(x)}$ is obtained from $\prec^{(x)}$ by simply replacing the clock x by its copy x'. Formally

$$\begin{array}{lll}\forall y, z \in X \setminus \{x\}, & y \,\overline{\prec}^{(x)}\, z & \text{if } y \prec^{(x)} z \\ \forall y \in X \setminus \{x\}, & y \,\overline{\prec}^{(x)}\, x' & \text{if } y \prec^{(x)} x \\ \forall y \in X \setminus \{x\}, & x' \,\overline{\prec}^{(x)}\, y & \text{if } x \prec^{(x)} y\end{array}$$

We then define $\overline{\prec}$ as the union of all the $\overline{\prec}^{(x)}$. It is clear that $\overline{\prec}$ is still a preorder on $X \cup X'$. Now, $\prec'$ can be obtained from $\overline{\prec}$ by first restricting it to $X' \times X'$ and then transforming each clock x' into its copy x. And we thus get that $w' \in \alpha'$.

We thus have proven that if $\alpha' \in \widehat{up}(\alpha)$, then for any valuation $w \in \alpha$, there exists a valuation $w' \in up(w) \cap \alpha'$. Condition $(\star\star\star)$ is thus satisfied. $\square$

From Theorem 3 and propositions 6 and 7, we get immediately the next theorem which is our main (effective) result concerning decidability of **diagonal-free** updatable automata.

Theorem 4. *Let*
- *$\mathcal{C} \subseteq \mathcal{C}_{df}(X)$ be a finite set of diagonal-free clock constraints,*
- *for any clock x, c_x be an integer constant such that, for any constraint $x \sim c$ of $\mathcal{C}$, it holds $c \leq c_x$,*
- *$\mathcal{U} \subseteq \mathcal{U}_{(c_x)_{x\in X}}$ be a finite set of updates.*

*Then the class **Uta**$(\mathcal{C},\mathcal{U})$ is decidable.*

This theorem is not yet sufficient for deciding, given an arbitrary (diagonal-free) timed automaton $\mathcal{A}$, whether its emptiness can be decided using a region automaton construction. If we can find constants $(c_x)_{x\in X}$ such that any update used in $\mathcal{A}$ is in $\mathcal{U}_{(c_x)_{x\in X}}$ and any constraint $x \sim c$ used in $\mathcal{A}$ satisfies $c \leq c_x$, then the emptiness of $\mathcal{A}$ can be checked using a region automaton construction. We finally now describe a procedure which gives a sufficient condition for the existence of such constants $(c_x)_{x\in X}$.

Let $\mathcal{C} \subseteq \mathcal{C}_{df}(X)$ be a set of diagonal-free clock constraints and let $\mathcal{U} \subseteq \mathcal{U}(X)$ be a set of updates such that

$$up = \bigwedge_{x\in X} up_x \in \mathcal{U} \implies \text{for all } x,\ up_x \in \{det_x, inf_x, sup_x, int_x\} \text{ where:} \qquad (\diamondsuit_{df})$$

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

If the Diophantine system of linear inequations on variables $(c_x)_{x\in X}$

$$\{c \leq c_x \mid x \sim c \in \mathcal{C} \text{ or } x :\sim c \in \mathcal{U}\} \cup \{c_z \leq c_y + c \mid z :\sim y + c \in \mathcal{U}\} \qquad (\mathcal{S}_{df})$$

has a solution, then $\mathcal{U} \subseteq \mathcal{U}_{(c_x)_{x\in X}}$ and $\mathcal{C}$ is compatible with $\mathcal{R}_{(c_x)_{x\in X}}$, and therefore, applying Theorem 4, the class ***Uta***$(\mathcal{C},\mathcal{U})$ of updatable timed automata is decidable.

Note that if all the constants c appearing in the updates $x :\sim y + c$ are positive, then the system $(\mathcal{S}_{df})$ always has a solution. Otherwise, from the results of [Dom91], the existence of a solution is decidable.

Remark 5. We have shown in section 4 that updates of the form $z := z - 1$ lead to an undecidable class of automata, whatever are the types of constraints used in the automata. Note that, fortunately, this is not in contradiction with the results above. Indeed, when dealing with such updates, the Diophantine system $(\mathcal{S}_{df})$ contains inequations of the form $c_z \leq c_z - 1$ and has therefore no solution.

Complexity. As for timed automata (see [AD94]), decidability of emptiness for a class of updatable timed automata verifying hypotheses of Theorem 4 is a PSPACE-complete problem; and the proof is quite similar.

Recall that for classical (untimed) automata (accepting finite or infinite sequences), decidability of the emptiness is NLOGSPACE-complete. The non deterministic on-the-fly algorithm consists in starting from an initial state q_0, to guess a new state q and to verify whether there is a transition from q_0 to q, which can be done without any additional space (just looking at the automaton). The algorithm continues by guessing a new state q' and by verifying the existence of a transition between q and q', and so on until a final state is reached. Therefore, besides the automaton, only two states have to be stored. Since a state can be coded in logarithmic space, we get that the emptiness problem is in NLOGSPACE (the proof of completeness can be found in any book on Complexity Theory).

Let now $\mathcal{A}$ be an updatable timed automaton in some class ***Uta***$(\mathcal{C},\mathcal{U})$ and $\mathcal{R}$ be a set of regions satisfying the hypotheses of Theorem 4. As explained, the emptiness of $L(\mathcal{A})$ can be checked by testing the emptiness on the untimed region automaton $\Gamma_{\mathcal{R}}(\mathcal{A})$. If we apply the algorithm recalled above and if we want to compute its complexity, we have to compute the space needed to encode a state of $\Gamma_{\mathcal{R}}(\mathcal{A})$. Such a state is a pair (q, R) where q is a (discrete) state of $\mathcal{A}$ and R a region of $\mathcal{R}_{(c_x)_{x\in X}}$. For encoding a region, it is sufficient to store, for each clock, two integers (the bounds of the interval where the clock is supposed to be) and, for each pair of clocks, a boolean which indicates whether the first clock is before the second in the preorder defining the region, or not.
Therefore, a state of $\Gamma_{\mathcal{R}}(\mathcal{A})$ can be encoded in polynomial space and emptiness of updatable timed automata, when belonging to a decidable class as described previously, is in PSPACE. Since these decidable classes contain in particular Alur and Dill's timed automata, we get immediately the PSPACE-hardness and thus the PSPACE-completeness.

5.3 Decidable Classes of General Updatable Timed Automata

We now investigate classes of updatable timed automata where general constraints are used. As we have noticed just after proposition 6, diagonal constraints are not compatible with sets of regions defined in the previous subsection. For example, if we deal with two clocks x and y, the region $x > 3 \wedge y > 2$ is neither included in $x - y \leq 1$, nor in $x - y > 1$. We have thus to define new sets of regions.

To this purpose we consider for each pair of clocks (y, z) in X an integer constant $d_{y,z}$ and we define the set

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

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

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

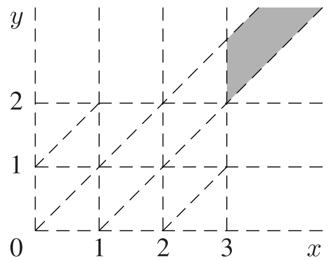
is the following subset of $\mathbb{T}^X$:

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

The finite set $\mathcal{R}_{(c_x)_{x\in X},(d_{y,z})_{y,z\in X}}$ of all such regions forms a partition of $\mathbb{T}^X$. By a proof very similar to the one of lemma 2, it is easy to verify that this set of regions also satisfies condition ($\star\star$), *i.e.* that the following lemma holds:

Lemma 5. *The set $\mathcal{R}_{(c_x)_{x\in X},(d_{y,z})_{y,z\in X}}$ is a set of regions.*

Example 7. 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 $]-1; 0[$).

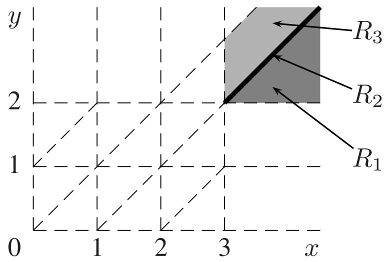


Once again, the compatibility of this set of regions with sets of clock constraints is easy and immediate.

Proposition 8. *Let $\mathcal{C} \subseteq \mathcal{C}(X)$ be such that for any clock constraint $x \sim c$ of $\mathcal{C}$, we have $c \leq c_x$ and for any clock constraint $x - y \sim c$ in $\mathcal{C}$, we have $-d_{y,x} \leq c \leq d_{x,y}$. Then the set of regions $\mathcal{R}_{(c_x)_{x\in X},(d_{y,z})_{y,z\in X}}$ is compatible with $\mathcal{C}$.*

As in the diagonal-free case, we now introduce a set of updates which depends on the constants $(c_x)_{x\in X}$ and $(d_{y,z})_{y,z\in X}$. They will be defined in such a way that they will be compatible with the set of regions we have just defined. Note that from the undecidability results of section 4, we have to restrict drastically the set of updates we use if we want to preserve the decidability.

Example 8. For example, if we consider the incrementation update $y := y+1$ and the set of regions depicted on the figure beside, the images of the region R_1 are the regions R_1, R_2 and R_3. But we can not reach region R_1 (resp. R_2, resp. R_3) from every point of region R_1. Thus, this set of regions is not compatible with the update $y := y + 1$.



Definition 2. *Let $(c_x)_{x\in X}$, $(d_{y,z})_{y,z\in X}$ be integer constants. The set $\mathcal{U}_{(c_x)_{x\in X},(d_{y,z})_{y,z\in X}}$ of local updates consists of the updates of the form $up = \bigwedge_{x\in X} up_x$ where, for each clock $x \in X$, up_x is a local update of one of the following forms:*

- *$x :\lhd c$ with $\lhd \in \{=, <, \leq\}$, $z \in X$, $c \in \mathbb{N}$, $c \leq c_x$ and, for any clock y, $c_y \geq c + d_{y,x}$*
- *$x := y$ with $y \in X$, and $c_x \leq c_y$ and, for any clock z, $d_{z,x} \leq d_{z,y}$, $d_{x,z} \leq d_{y,z}$*

As claimed by the following proposition, this set of updates and the set of regions previously defined are suitable for handling updatable timed automata with general clock constraints.

Proposition 9. *Let $(c_x)_{x\in X}$, $(d_{y,z})_{y,z\in X}$ be integer constants. Then the set of regions $\mathcal{R}_{(c_x)_{x\in X},(d_{y,z})_{y,z\in X}}$ is compatible with the set of updates $\mathcal{U}_{(c_x)_{x\in X},(d_{y,z})_{y,z\in X}}$.*

Proof. As in the case of diagonal-free updatable timed automata, we first deal with the particular case of simple updates.

Assume that $\alpha = ((I_x)_{x\in X}, (J_{x,y})_{x,y\in X}, \prec)$ where $\prec$ is a total preorder on X_0 and assume also that up is a simple update over z, then the region $\alpha' = ((I'_x)_{x\in X}, (J'_{x,y})_{x,y\in X}, \prec')$ (where $\prec'$ is a total preorder on X'_0) is in $\widehat{up}(\alpha)$ if and only if $I'_x = I_x$ for all $x \neq z$, $J'_{x,y} = J_{x,y}$ for all $x, y \neq z$ and:

if *up* is *z:* $\sim$ *c*, I'_z can be any interval of $\mathcal{J}_z$ which intersects $\{\gamma \in \mathbb{T} \mid \gamma \sim c\}$ and

- either I'_z is of the form $[d]$ and thus
  - $X'_0 = X_0 \setminus \{z\}$
  - $\prec' = \prec \cap (X'_0 \times X'_0)$
  - $X'_\infty = \{(x,y) \in X_\infty \mid (x \neq z \wedge y \neq z)$ or $(x = z \wedge I_y =]c_y;\infty[)$ or $(I_x =]c_x;\infty[\wedge y = z)$ and $\forall (x,y) \in X'_\infty$,
    - $J'_{x,y} = J_{x,y}$ if $x \neq z$ and $y \neq z$
    - $J'_{x,z} =]d_{x,z};\infty[$.
      Note that if v is a valuation such that $c_x < v(x)$ and $v(z) \lhd c$ with $\lhd \in \{=,<,\leq\}$, then $c_x - c < v(x) - v(z)$. Thus, from the hypothesis $c_x \geq c + d_{x,z}$, we get $d_{x,z} < v(x) - v(z)$.
    - $J'_{z,y} =]-\infty;-d_{z,y}[$.
      Note that if v is a valuation such that $c_y < v(y)$ and $v(z) \lhd c$ with $\lhd \in \{=,<,\leq\}$, then $v(z) - v(y) < c - c_y$. Thus, from the hypothesis $c_y \geq c + d_{z,y}$, we get $v(z) - v(y) < -d_{z,y}$.
- either I'_z is of the form $]d;d+1[$ and thus
  - $X'_0 = X_0 \cup \{z\}$
  - $\prec'$ is any total preorder on X'_0 which coincides with $\prec$ on $X'_0 \setminus \{z\}$
  - $X'_\infty = \{(x,y) \in X_\infty \mid (x \neq z \wedge y \neq z)$ or $(x = z \wedge I_y =]c_y;\infty[)$ or $(I_x =]c_x;\infty[\wedge y = z)$ and $\forall (x,y) \in X'_\infty$,
    - $J'_{x,y} = J_{x,y}$ if $x \neq z$ and $y \neq z$
    - $J'_{x,z} =]d_{x,z};\infty[$.
    - $J'_{z,y} =]-\infty;-d_{z,y}[$.

if *up* is *z:* $\sim$ *y*, let us first define I'_z.

- if $I_y = [d]$, $I'_z = [d]$ if $d \leq c_z$, $I'_z =]c_z;\infty[$ otherwise
- if $I_y =]d;d+1[$, $I'_z =]d;d+1[$ if $d < c_z$, $I'_z =]c_z;\infty[$ otherwise
- if $I_y =]c_y;\infty[$, $I'_z =]c_z;\infty[$ (since by hypothesis $c_z \leq c_y$)

Now

- either I'_z is of the form $[d]$ (and thus $I_y = [d]$ from what precedes)
  - $X'_0 = X_0 \cup \{z\}$
  - $\prec' = \prec \cap (X'_0 \times X'_0)$
  - $X'_\infty = \{(x,x') \in X_\infty \mid (x \neq z \wedge x' \neq z)$ or $(x = z \wedge I_{x'} =]c_{x'};\infty[)$ or $(I_x =]c_x;\infty[\wedge x' = z)\}$
    and $\forall (x,x') \in X'_\infty$,
    - $J'_{x,x'} = J_{x,x'}$ if $x \neq z$ and $x' \neq z$
    - $J'_{z,x'}$ is the unique interval of $\mathcal{J}_{z,x'}$ which contains $J_{z,x'}$.
      Note that unicity comes from the hypothesis that $d_{z,x'} \leq d_{y,x'}$
    - $J'_{x,z}$ is the unique interval of $\mathcal{J}_{x,z}$ which contains $J_{x,z}$.
      Note that unicity comes from the hypothesis that $d_{x,z} \leq d_{x,y}$

– either I'_z is of the form $]d; d+1[$ (and thus $I_y =]d; d+1[$, too)

- $X'_0 = X_0 \cup \{z\}$
- $\prec'$ is any total preorder on X'_0 which coincides with $\prec$ on $X'_0 \setminus \{z\}$ and such that $z \prec' y$ and $y \prec' z$
- The set X'_∞ and the intervals $J'_{x,x'}$ are defined as in the previous case $I'_z = [d]$

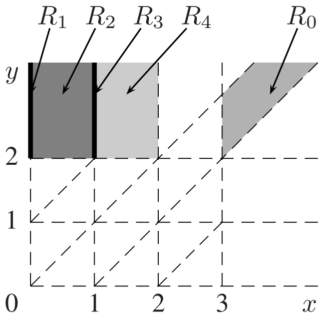
– either I'_z is of the form $]c_z; \infty[$

- $X'_0 = X_0 \setminus \{z\}$
- $\prec' = \prec \cap (X'_0 \times X'_0)$
- $X'_\infty = X_\infty \cup \{(x,z),(z,x) \mid x \in X\}$ and $J'_{x,x'} = J_{x,x'}$ if $x \neq z$ and $x' \neq z$. The computation of $J'_{z,x}$ (and $J'_{x,z}$) requires to distinguish several cases depending of the form of I_x and I_y

1. $I_x = [f]$, $I_y = [g]$. Then
$$J'_{z,x} = \begin{cases} [g-f] & \text{if } -d_{x,z} \leq g-f \leq d_{z,x} \\ ]d_{z,x}; \infty[& \text{if } d_{z,x} < g-f \\ ]-\infty; -d_{x,z}[& \text{if } g-f < -d_{x,z} \end{cases}$$
2. $I_x = [f]$, $I_y =]g; g+1[$. Then
$$J'_{z,x} = \begin{cases}]g-f-1; g-f[& \text{if } -d_{x,z} \leq g-f-1 < d_{z,x} \\ ]d_{z,x}; \infty[& \text{if } d_{z,x} \leq g-f-1 \\ ]-\infty; -d_{x,z}[& \text{if } g-f-1 < -d_{x,z} \end{cases}$$
3. $I_x = [f]$, $I_y =]c_y; \infty[$. Then
$J'_{z,x}$ is the unique interval of $\mathcal{J}_{z,x}$ which contains $J_{y,x}$.
Note that unicity comes from the hypothesis that $d_{z,x} \leq d_{y,x}$ and $d_{x,z} \leq d_{x,y}$
4. $I_x =]f; f+1[$, $I_y = [g]$.
This case is identical to case 2 above.
5. $I_x =]f; f+1[$, $I_y =]g; g+1[$. Then
$$J'_{z,x} = \begin{cases} \text{If } x \prec y \wedge y \prec x \text{ then} & [g-f] & \text{when } -d_{x,z} \leq g-f \leq d_{z,x} \\ &]d_{z,x}; \infty[& \text{when } d_{z,x} < g-f \\ &]-\infty; -d_{x,z}[& \text{when } g-f < -d_{x,z} \\ \text{If } x \prec y \wedge y \not\prec x \text{ then} &]g-f; g-f+1[& \text{when } -d_{x,z} \leq g-f < d_{z,x} \\ &]d_{z,x}; \infty[& \text{when } d_{z,x} \leq g-f \\ &]-\infty; -d_{x,z}[& \text{when } g-f < -d_{x,z} \\ \text{If } x \not\prec y \wedge y \prec x \text{ then} &]g-f-1; g-f[& \text{when } -d_{x,z} \leq g-f-1 < d_{z,x} \\ &]d_{z,x}; \infty[& \text{when } d_{z,x} \leq g-f-1 \\ &]-\infty; -d_{x,z}[& \text{when } g-f-1 < -d_{x,z} \end{cases}$$
6. $I_x =]f; f+1[$, $I_y =]c_y; \infty[$. This case is identical to case 3 above.
7. $I_x =]c_x; \infty[$. This case is identical to case 3 above.

From this construction, it is easy to prove, in a similar way than for lemma 3, that condition $(\star\star\star)$ holds for simple updates.

The extension to local updates of $\mathcal{U} \subseteq \mathcal{U}_{(c_x)_{x \in X},(d_{y,z})_{y,z \in X}}$ (under the hypotheses of the proposition) is obtained by a technique similar to the one used in proposition 7. $\square$



Example 9. Consider the regions depicted on the left. We want to compute the updating successors of the region R_0 by the update $x :< 2$. The four updating successors are drawn on the figure. Their equations are:

- Region R_1: $I'_x = [0]$ and $I'_y =]2; +\infty[$
- Region R_2: $I'_x =]0; 1[$, $I'_y =]2; +\infty[$ and $J_{y,x} =]1; +\infty[$
- Region R_3: $I'_x = [1]$ and $I'_y =]2; +\infty[$
- Region R_4: $I'_x =]1; 2[$, $I'_y =]2; +\infty[$ and $J_{y,x} =]1; +\infty[$

Our main effective result concerning the decidability of general updatable automata is given by the following theorem. Its proof follows immediately from Theorem 3 and propositions 8 and 9.

Theorem 5. *Let $\mathcal{C} \subseteq \mathcal{C}(X)$ be a finite set of general clock constraints such that:*

- *for every clock x, a constant c_x such that for any constraint $x \sim c$ in $\mathcal{C}$, $c \leq c_x$,*
- *for every pair of clocks (x, y), a constant $d_{x,y}$ such that for any constraint $x - y \sim c$ in $\mathcal{C}$, $c \leq d_{x,y}$,*

*and let $\mathcal{U} \subseteq \mathcal{U}_{(c_x)_{x\in X},(d_{x,y})_{x,y\in X}}$ be a set of updates. The class **Uta**$(\mathcal{C}, \mathcal{U})$ is then decidable.*

Like for Theorem 4, if we want to apply the previous theorem to a given updatable timed automaton $\mathcal{A}$, we need to find (if they exist) some constants $(c_x)_{x\in X}$ and $(d_{x,y})_{x,y\in X}$ for which the updates and constraints of $\mathcal{A}$ satisfy the hypothesis of this theorem. Let us now describe a procedure which ensures the existence of such constraints.

Let $\mathcal{C} \subseteq \mathcal{C}(X)$ be a finite set of arbitrary constraints and let $\mathcal{U} \subseteq \mathcal{U}(X)$ be a finite set of updates such that:

$$up = \bigwedge_{x\in X} up_x \in \mathcal{U} \implies \forall x \in X,\ up_x \in \begin{array}{l}\{x := c, x :< c, x :\leq c \mid c \in \mathbb{N}\}\\ \cup\ \{x := y \mid y \in X\}\end{array} \qquad (\diamondsuit_{gen})$$

If the Diophantine system of linear inequations on the variables $(c_x)_{x\in X}$ and $(d_{x,y})_{x,y\in X}$

$$\begin{array}{ll} & \{c \leq \max_x \mid x \sim c \in \mathcal{C}\} \\ \cup & \{c \leq \max_{x,y} \mid x - y \sim c \in \mathcal{C}\} \\ \cup & \{c \leq \max_x,\ \max_z \geq c + \max_{z,x} \mid x :< c \text{ or } x :\leq c \text{ or } x := c\ \in \mathcal{U},\ \text{and } z \in X\} \\ \cup & \{\max_x \leq \max_y,\ \max_{z,y} \geq \max_{z,x},\ \max_{x,z} \leq \max_{y,z} \mid x := y \in \mathcal{U} \text{ and } z \in X\} \end{array} \qquad (\mathcal{S}_{gen})$$

has a solution, then $\mathcal{U} \subseteq \mathcal{U}_{(c_x)_{x\in X},(d_{x,y})_{x,y\in X}}$ and $\mathcal{C}$ is compatible with $\mathcal{R}_{(c_x)_{x\in X},(d_{x,y})_{x,y\in X}}$. And thus, from Theorem 5, the class ***Uta***$(\mathcal{C}, \mathcal{U})$ is decidable.

It is easy to verify that the system ($\mathcal{S}_{gen}$) always has a solution. We thus get the following theorem:

Theorem 6. *Let $\mathcal{C} \subseteq \mathcal{C}(X)$ be a finite set of arbitrary constraints and let $\mathcal{U}$ be a finite set of updates defined as in ($\diamondsuit_{gen}$). Then the class **Uta**$(\mathcal{C}, \mathcal{U})$ of updatable timed automata is decidable.*

Remark 6. From the undecidability results of the previous section, this theorem is the most general we can expect when dealing with general clock constraints. Nevertheless, under precise conditions, we could refine the results and exhibit decidable subclasses which use updates not of the form ($\diamondsuit_{gen}$). For instance, let $(c_x)_{x\in X}$, $(d_{y,z})_{y,z} \in X$ be constants. The set of regions $\mathcal{R}_{(c_x)_{x\in X},(d_{y,z})_{y,z\in X}}$ is compatible with, for examples, updates like:

- $z := y + c$ as soon as $c_z \leq c_y + c$ and for each clock x, $d_{x,z} \leq d_{x,y} - c$ and $d_{z,x} \leq d_{y,x} + c$
- $z :> c$ as soon as $c \leq c_z$ and for each clock x, $c_z - c_x \geq d_{z,x}$

However, we will not give details of these refinements, if one is needed for a special model, then the previous proof can be extended.

Complexity. As in the diagonal-free case (see the end of section 5.2), emptiness for decidable classes of updatable timed automata with arbitrary clock constraints, as characterized in Theorem 5, is PSPACE-complete. Indeed, a region from a set of the form $\mathcal{R}_{(c_x)_{x\in X},(d_{y,z})_{y,z\in X}}$ can still be encoded in polynomial space.

5.4 Conclusion and Discussion

Table 2 summarizes the undecidability and decidability results obtained in the two previous sections. In order to have a global and readable picture, we do not recall the precise conditions on the constants given in the hypotheses of our two main theorems 4 and 5, under which decidability is ensured.

<table>
<tr><th></th><th>$\mathcal{U}_0(X) \cup ...$</th><th>Diagonal-free constraints</th><th>General constraints</th></tr>
<tr><td>1</td><td>$x := c, x := y$</td><td rowspan="3">PSPACE-complete</td><td>PSPACE-complete</td></tr>
<tr><td>2</td><td>$x := x + 1$</td><td rowspan="3">Undecidable</td></tr>
<tr><td>3</td><td>$x := y + c$</td></tr>
<tr><td>4</td><td>$x := x - 1$</td><td>Undecidable</td></tr>
<tr><td>5</td><td>$x :< c$</td><td rowspan="4">PSPACE-complete</td><td>PSPACE-complete</td></tr>
<tr><td>6</td><td>$x :> c$</td><td rowspan="4">Undecidable</td></tr>
<tr><td>7</td><td>$x :\sim y + c$</td></tr>
<tr><td>8</td><td>$y + c <: x :< y + d$</td></tr>
<tr><td>9</td><td>$y + c <: x :< z + d$</td><td>Undecidable</td></tr>
</table>

with $\sim \in \{\leq, <, >, \geq\}$ and $c,\ d \in \mathbb{Q}^+$

Table 2. Decidability results

It is worth to notice that, contrary to the case of Alur and Dill's timed automata, considering diagonal-free clock constraints or arbitrary clock constraints do not lead to similar decidability results.

Note also that differences between decidable and undecidable classes are sometimes tricky. Among these differences, let us mention for instance the following facts:

- when only diagonal-free clock constraints are used, decrementation leads to undecidable classes whereas incrementation leads to decidable classes (see lines 2 and 4)
- when arbitrary clock constraints are used, both decrementations and incrementations lead to undecidable classes (see also lines 2 and 4)
- non-deterministic updates of the form $x :< c$ always lead to decidable classes whereas updates of the form $x :> c$ lead to decidable classes only when diagonal-free clock constraints are used (see lines 5 and 6)
- non-deterministic updates of the form $x + c :< z :< y + d$ always lead to undecidable classes whereas updates of the form $y + c <: z :< y + d$ lead to decidable classes if diagonal-free clock constraints are used (see lines 8 and 9)

6 Expressiveness of Updatable Timed Automata

Now that we have described precisely the frontier between undecidability and decidability, it becomes natural and interesting to study the expressiveness of the decidable subclasses and compare them with the expressiveness of timed automata and timed automata with ε-transitions (or silent actions), as defined originally by Alur and Dill ([AD90,AD94], see section 3.1).

We start by defining some criteria to compare automata in section 6.1. We then prove that ε-transitions are unavoidable if we want to express the languages recognized by updatable timed automata using classical timed automata, see section 6.2. We then study the easier case of updatable automata using deterministic updates in section 6.3 and the general case in section 6.4.

6.1 Several Equivalence Relations

We recall in this section several known criteria to compare automata.

Language equivalence. The simplest criterium to compare automata is the equality of the accepted languages. Two timed automata are said *language equivalent* whenever they accept the same timed language. We extend this definition to families of timed automata ; two families of timed automata, say Aut_1 and Aut_2, are language equivalent whenever every timed automaton from one of the families is language equivalent to an automaton of the other family. We then write $Aut_1 \equiv_\ell Aut_2$.
For example, it is well known that diagonal constraints can be removed from timed automata without changing the expressiveness of the model (see Remark 1). With the formalism presented above, it can be written as

$$\textbf{\textit{Uta}}(\mathcal{C}_{df}(X),\mathcal{U}_0(X)) \equiv_\ell \textbf{\textit{Uta}}(\mathcal{C}(X),\mathcal{U}_0(X)) \ .$$

Transition systems and similarity. Language equivalence does not provide any information about the internal structure of the automata, contrary to similarity. To define similarity, we first need to recall the notion of transition systems.

Definition 3. *A* transition system *is a tuple $\mathcal{T} = (S, \Gamma, s_0, \longrightarrow)$ where S is a set of states, Γ is a finite or infinite alphabet, $s_0 \in S$ is the initial state and $\longrightarrow \subseteq S \times \Gamma \times S$ is a set of transitions.*

If $\mathcal{T}$ is such a transition system, an *execution* in $\mathcal{T}$ is a sequence of consecutive transitions

$$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \dots$$

where for every $i > 0$, $s_{i-1} \xrightarrow{\alpha_i} s_i$ is a transition of $\mathcal{T}$.

The *similarity* [Par81,Mil89] defines step to step a correspondance between two transition systems. A transition system $\mathcal{T} = (S, \Gamma, s_0, \longrightarrow)$ *simulates* a transition system $\mathcal{T}' = (S', \Gamma, s'_0, \longrightarrow')$ if there exists a relation $\succcurlyeq \subseteq S \times S'$ such that:

INITIALIZATION: $\forall s_0 \in S_0,\ \exists s'_0 \in S'_0$ s.t. $s_0 \succcurlyeq s'_0$

PROPAGATION: if $s_1 \succcurlyeq s'_1$ and $s_1 \xrightarrow{e} s_2$ then there exists $s'_2 \in S'$
(TRANSFER) s.t. $s'_1 \xrightarrow{e}{}' s'_2$ and $s_2 \succcurlyeq s'_2$

Such a relation is called a *simulation relation*. If the relation $\succcurlyeq^{-1}$ defined by

$$x \succcurlyeq^{-1} y \iff y \succcurlyeq x$$

is also a simulation relation, then $\succcurlyeq$ is a *bisimulation relation*.

Timed transition systems are particular transition systems where the alphabet contains actions corresponding to time elapsing.

Definition 4. *A* timed transition system *on the alphabet Σ and the time domain $\mathbb{T}$ is a transition system $\mathcal{T}(S, \Gamma, s_0, \longrightarrow)$ where Γ is the set $\Sigma \cup \{\varepsilon\} \cup \{\epsilon(d) \mid d \in \mathbb{T}\}$ and the transition $\longrightarrow$ satisfies the following properties:*

- TEMPORAL DETERMINISM: *for all the states* s, s', s'' *of* S *and for every* $d \in \mathbb{T}$*, if* $s \xrightarrow{\epsilon(d)} s'$ *and* $s \xrightarrow{\epsilon(d)} s''$*, then* $s' = s''$*.*
- TIME ADDITIVITY: *for all the states* s, s'' *of* S *and for all* $d_1, d_2 \in \mathbb{T}$*, if* $s \xrightarrow{\epsilon(d_1+d_2)} s''$*, then there exists* $s' \in S$ *such that* $s \xrightarrow{\epsilon(d_1)} s'$ *and* $s' \xrightarrow{\epsilon(d_2)} s''$*.*
- 0-DELAY: *for all the states* $s, s' \in S$*,* $s \xrightarrow{\epsilon(0)} s'$ *if and only if* $s = s'$*.*

The three conditions that we just described are classical when we consider process algebra like TCCS [Yi90,Yi91].

If $\mathcal{T}$ is such a timed transition system, a *delay execution* is an execution of the form

$$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \dots \xrightarrow{\alpha_n} s_n$$

such that $n \geq 0$, for every $1 \leq i \leq n$, $\alpha_i = \varepsilon$ or $\alpha_i = \epsilon(d_i)$ for some $d_i \in \mathbb{T}$.

If $\mathcal{T} = (S, \Gamma, s_0, \longrightarrow)$ is a timed transition system, we define the *abstract transition system* associated with $\mathcal{T}$ by $\mathcal{T}_{abs} = (S, \Gamma, s_0, \Longrightarrow)$ where

$$\begin{cases} s \overset{a}{\Longrightarrow} s' \text{ if } a \neq \varepsilon \text{ and there exists } s'' \in S,\ s \xrightarrow{\varepsilon}{}^{*} s'' \xrightarrow{a} s' \\ s \overset{\epsilon(d)}{\Longrightarrow} s' \text{ if } \begin{cases} \text{there exists a delay execution} \\ s = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \dots \xrightarrow{\alpha_n} s_n = s' \\ \text{such that } d = \sum\{d_i \mid \alpha_i = \epsilon(d_i)\} \end{cases} \end{cases}$$

where the relation $\xrightarrow{\varepsilon}{}^{*}$ represents the reflexive and transitive closure of $\xrightarrow{\varepsilon}$. The transition system $\mathcal{T}_{abs}$ abstracts silent actions of $\mathcal{T}$. The relation $\xrightarrow{\varepsilon}{}^{*}$ thus corresponds to $\overset{\epsilon(0)}{\Longrightarrow}$. Note also that the relation $\overset{a}{\Longrightarrow}$ only abstracts silent actions that can be done before action a.

As a timed transition system is a particular transition system, the notion of similarity defined before can be applied.

Strong and weak (bi)similarity. An updatable timed automaton $\mathcal{A}(Q, X, \Sigma_\varepsilon, I, F, R, T)$ defines in a natural way two timed transition systems:

- the transition system $\mathcal{T}(\mathcal{A}) = (Q \times \mathbb{T}^X, \Sigma_\varepsilon, \mathbb{T}, (q_0, \mathbf{0}), \longrightarrow)$ where the transition relation $\longrightarrow$ is defined by:

$$\begin{cases} (q, v) \xrightarrow{\epsilon(d)} (q, v + d) \\ (q, v) \xrightarrow{a} (q', v') \text{ if there exists } q \xrightarrow{\varphi, a, up} q' \in T \text{ s.t. } v \models \varphi \text{ and } v' \in up(v) \end{cases}$$

- the abstract transition system $\mathcal{T}_{abs}(\mathcal{A})$ defined as previously from $\mathcal{T}(\mathcal{A})$.

Of course, if $\mathcal{A}$ is a timed automaton without silent actions, $\mathcal{T}(\mathcal{A})$ and $\mathcal{T}_{abs}(\mathcal{A})$ are identical.

An updatable timed automaton $\mathcal{A}$ *strongly simulates* an other updatable timed automaton $\mathcal{B}$, and we will note $\mathcal{A} \succcurlyeq_s \mathcal{B}$, whenever $\mathcal{T}(\mathcal{A})$ simulates $\mathcal{T}(\mathcal{B})$. We say that $\mathcal{A}$ and $\mathcal{B}$ are *strongly bisimilar*, and we will note $\mathcal{A} \equiv_s \mathcal{B}$, whenever there exists a bisimulation relation $\equiv$ such that $\mathcal{T}(\mathcal{A}) \equiv \mathcal{T}(\mathcal{B})$.

An updatable timed automaton $\mathcal{A}$ *weakly simulates*[7] another updatable timed automaton $\mathcal{B}$, and we will note $\mathcal{A} \succcurlyeq_w \mathcal{B}$, whenever $\mathcal{T}_{abs}(\mathcal{A})$ simulates $\mathcal{T}_{abs}(\mathcal{B})$. We say that $\mathcal{A}$ and $\mathcal{B}$ are *weakly bisimilar*, and we note $\mathcal{A} \equiv_w \mathcal{B}$, whenever there exists a bisimulation relation $\equiv$ such that $\mathcal{T}_{abs}(\mathcal{A}) \equiv \mathcal{T}_{abs}(\mathcal{B})$.

[7] Note that this definition of weak simulation is quite different from the usual one because, as said before, the transition relation $\overset{a}{\Longrightarrow}$ only abstracts silent actions that can be done before the other actions, whereas, in the classical definition, the transition relation abstracts all the silent actions, *i.e.* those that can be done before or after the real actions.

Remark 7. Of course, two strongly bisimilar updatable timed automata are also weakly bisimilar. If a bisimulation relation preserves the final and repeated states, two strongly or weakly bisimilar automata are language equivalent.

We close these preliminaries by a technical result ensuring that we can restrict our study to updatable timed automata where all constants appearing in the constraints or in the updates are integer.
Let $\mathcal{A}$ be an updatable timed automaton and λ a constant. We denote by $\lambda\mathcal{A}$ the timed automaton in which all the constants appearing in the constraints or the updates of $\mathcal{A}$ are multiplied by λ. The proof of the following lemma follows the one of lemma 4.1 page 15 in [AD94] which claims a similar result for language equivalence within timed automata.

Lemma 6. *Let $\mathcal{A}$ and $\mathcal{B}$ be two timed automata and $\lambda \in \mathbb{Q}^{+*}$ a constant. Then*

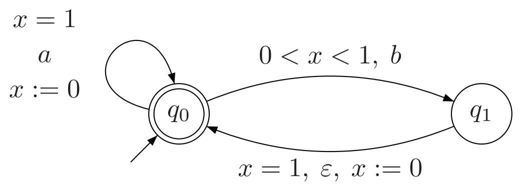
$$\mathcal{A} \succcurlyeq_w \mathcal{B} \iff \lambda\mathcal{A} \succcurlyeq_w \lambda\mathcal{B} \quad \textit{and} \quad \mathcal{A} \succcurlyeq_s \mathcal{B} \iff \lambda\mathcal{A} \succcurlyeq_s \lambda\mathcal{B}$$

Hence, in the rest of this section, we may assume that only integer constants are used.

We have now all the comparison tools that will be useful in our next study of the expressiveness of decidable subclasses of updatable timed automata.

6.2 ε-Transitions are Necessary

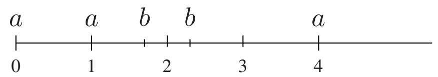
We first prove that ε-transitions are necessary to express the decidable fragment of updatable timed automata described in section 5. Let us consider the timed automaton $\mathcal{A}$ with silent actions described by the following picture:



There is no classical timed automaton without silent action accepting the same timed language as $\mathcal{A}$ [BDGP98]. We will prove that there exists an updatable timed automaton with general constraints and updates of the form $x := c$ or $x :< c$ (c integer) which recognizes the timed language $L(\mathcal{A})$. This timed language can be described by:

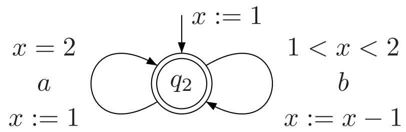
$$(a_i, t_i)_{i \geq 1} \in L \iff \forall i \geq 1, \begin{cases} t_i = i \text{ and } a_i = a \\ \text{or} \\ t_i \in]i-1; i[\text{ and } a_i = b \end{cases}$$

An execution in this automaton can thus be represented by the following scheme:



expressing that a actions can be performed each time unit, but not if a b has been performed during the last unit of time.

This timed language is recognized by the updatable timed automaton $\mathcal{B}$ on the following picture:

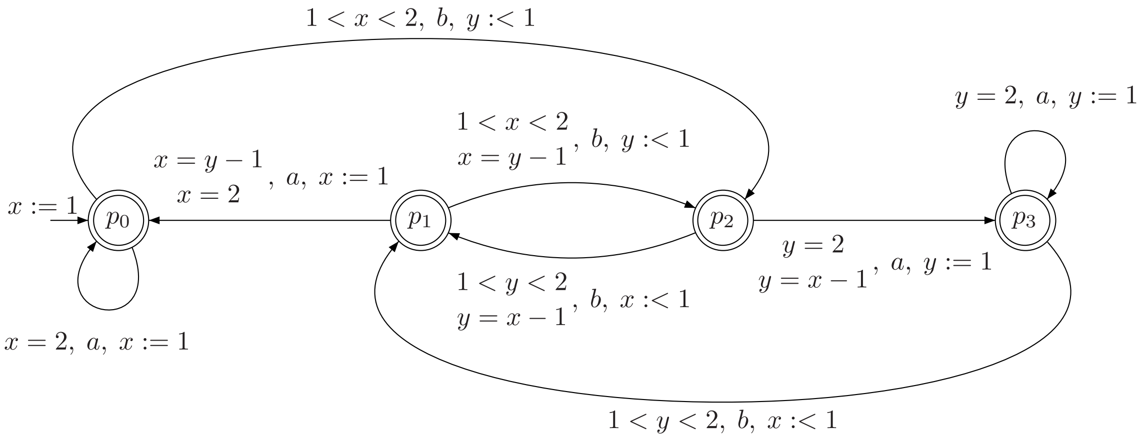


where the clock x is set to 1 when first entering state q_2.
By considering for example the bisimulation relation

$$\mathcal{R} = \Big\{((q_0, v), (q_2, v + 1)) \mid v \in \mathbb{T}^{\{x\}}\Big\} \cup \Big\{((q_1, v), (q_2, v)) \mid v \in \mathbb{T}^{\{x\}}\Big\}$$

it is easy to see that $\mathcal{A}$ and $\mathcal{B}$ are weakly bisimilar, and thus $L(\mathcal{A}) = L(\mathcal{B})$.

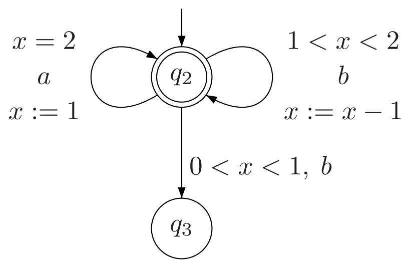
In section 4, we noticed that adding the decrementation of clocks to the classical model leads in general to undecidability. However, in this precise case, clock x is bounded by 2, we will thus be able to transform automaton $\mathcal{B}$ into an updatable timed automaton belonging to some decidable class as described in section 5. Let us indeed consider the following automaton $\mathcal{D}$:



Claim: $\mathcal{D}$ recognizes precisely the timed language $L(\mathcal{A}) = L(\mathcal{B})$.

Proof. We start by describing in an informal manner how $\mathcal{D}$ behaves. A state p_0 or p_3 can be reached only if an a has just been performed and a state p_1 or p_2 can be reached only if a b has just been performed. The values of x and y are both 1 when reaching state p_0 or p_3 (an easy verification can be done by analyzing the transitions arriving in these states). From any of these two states, a sequence of a's, one at each time unit, can be performed. Moreover, state p_1 or p_2 can be reached when an action b is performed, before one time unit has passed.

To prove that $L(\mathcal{B}) = L(\mathcal{D})$, we transform the automaton $\mathcal{B}$ in the following way. We first add a "hole" (state q_3) with a unique transition leading to q_3, namely the transition $q_2 \xrightarrow{0<x<1,\ b} q_3$. We denote by $\mathcal{B}_m$ the resulting automaton. It can be depicted as:



We then define the relation $\mathcal{R}'$ by:

$$\begin{aligned}\mathcal{R}' = \ & \{(q_2,\alpha),(p_0,(\alpha+1,\alpha+1)) \mid 0\le\alpha\le 1\} \cup \{(q_2,\alpha),(p_3,(\alpha+1,\alpha+1)) \mid 0\le\alpha\le 1\}\\ & \cup \{((q_2,\alpha),(p_2,(\alpha+1,\alpha))) \mid 0<\alpha\le 1\} \cup \{((q_2,\alpha),(p_1,(\alpha,\alpha+1))) \mid 0<\alpha\le 1\}\\ & \cup \{((q_3,\alpha),(p_2,(\beta,\alpha))) \mid \alpha>0 \text{ and } \beta\neq\alpha+1\} \cup \{((q_3,\alpha),(p_2,(\alpha,\beta))) \mid \alpha>0 \text{ and } \beta\neq\alpha+1\}\end{aligned}$$

The transfer property is satisfied in a trivial way. The relation $\mathcal{R}'$ is thus a bisimulation relation and the automata $\mathcal{D}$ and $\mathcal{B}_m$ are bisimilar. Moreover, $\mathcal{B}$ and $\mathcal{B}_m$ obviously recognize the same timed language. □

We thus get the following theorem:

Theorem 7. *The decidable subclass of updatable timed automata which use general clock constraints (as described in Section 5.3) is strictly more expressive (for the language equivalence $\equiv_\ell$) than classical timed automata without ε-transitions.*

6.3 Expressiveness of Deterministic Updates

We start our expressiveness study by considering deterministic updates only. Recall that these updates, defined in section 2.3, are built using simple updates of one of the following form:

1. $x := c$ with $x \in X$ and $c \in \mathbb{N}$
2. $x := y$ with $x, y \in X$
3. $x := y + c$ with $x, y \in X$ and $c \in \mathbb{Z} \setminus \{0\}$

Recall that thanks to Lemma 6, we assume, without loss of generality, that constants are in $\mathbb{N}$ and $\mathbb{Z}$ (we do not need to consider constants in $\mathbb{Q}$).
In a first step, we consider simple updates of one of the forms 1 or 2. The fact that updatable timed automata using such updates and classical timed automata are language equivalent is often considered as a **"folklore"** result. However, we did not find any proof of this result in the literature. Hence, and for the sake of completeness, we propose a complete proof.

If U is a set of simple deterministic updates, we denote by $\boldsymbol{Lu}(U)$ the set of updates which can be written as $\bigwedge_{x\in X} up_x$ where $up_x \in U$ for every $x \in X$.

Theorem 8. *Let $\mathcal{U} \subseteq \boldsymbol{Lu}(\{x := d \mid x \in X \text{ and } d \in \mathbb{N}\} \cup \{x := y \mid x,\ y \in X\})$ be a set of updates. Let $\mathcal{A} \in \boldsymbol{Uta}(\mathcal{C}(X),\mathcal{U})$ (resp. $\mathcal{A} \in \boldsymbol{Uta}_\varepsilon(\mathcal{C}(X),\mathcal{U})$). There exists a timed automaton $\mathcal{B} \in \boldsymbol{Uta}(\mathcal{C}(X),\mathcal{U}_{cst}(X))$ (resp. $\mathcal{B} \in \boldsymbol{Uta}_\varepsilon(\mathcal{C}(X),\mathcal{U}_{cst}(X))$) such that $\mathcal{A} \equiv_s \mathcal{B}$.*

Remind (see section 2.3) that $\mathcal{U}_{\text{cst}}(X)$ denotes updates to constants, that is updates of the form $x := c$.

Proof. Let $\mathcal{A} = (Q, X, \Sigma, I, F, R, T)$ be a timed automaton in $\boldsymbol{Uta}(\mathcal{C}(X),\mathcal{U})$. We construct a timed automaton $\mathcal{B} = (Q', X, \Sigma, I', F', R', T')$ in $\boldsymbol{Uta}(\mathcal{C}(X),\mathcal{U}_{\text{cst}}(X))$ such that $\mathcal{A} \equiv_s \mathcal{B}$.

Assume that $X = \{x_1, \ldots, x_n\}$. We set:

- $Q' = Q \times X^X$,
- $I' = I \times \{\mathsf{Id}\}$ where Id is the identity of X,
- $F' = F \times X^X$
- $R' = R \times X^X$.

Intuitively, in a state (q, σ) (with $q \in Q$ and $\sigma \in X^X$), the value of clock x is stored in the clock $\sigma(x)$. We now just have to define the set of transitions T' of $\mathcal{B}$.

Let us consider a transition $q \xrightarrow{\varphi,a,up} q'$ of $\mathcal{A}$ and a state (q, σ) of $\mathcal{B}$. We associate the function $\overline{up} : X \longrightarrow X \cup \mathbb{N}$ to up, where $\overline{up}(x)$ is:

- d whenever $x := d$ is part of the update up,
- y whenever $x := y$ is part of the update up,
- x in all other cases (the update is thus implicitely $x := x$).

In $\mathcal{B}$, there will be a transition

$$(q, \sigma) \xrightarrow{\varphi', a, up'} (q', \sigma')$$

such that:

- If $\overline{up}(x) \in X$, then $\sigma'(x) = \sigma \circ \overline{up}(x)$. If $\overline{up}(x) \notin X$, it is a bit more complicated. Some clocks are not used (it means that they do not correspond to any of the $\sigma'(x)$ already defined). We choose some of these clocks in order to define the $\sigma'(x)$ which are not already defined, *i.e.* the $\sigma'(x)$ such that $\overline{up}(x) \notin X$. More formally, we have:

$$\#\{x \in X \mid \overline{up}(x) \in X\} \geq \#\{\overline{up}(x) \mid x \in X \text{ and } \overline{up}(x) \in X\}$$

  and thus

$$\#\{x \in X \mid \overline{up}(x) \notin X\} \leq \#\left(X \setminus \{\overline{up}(x) \mid x \in X \text{ and } \overline{up}(x) \in X\}\right)$$

  We can thus consider an injective application ι defined on the set $\{x \in X \mid \overline{up}(x) \notin X\}$ onto the set $X \setminus \{\overline{up}(x) \mid x \in X \text{ and } \overline{up}(x) \in X\}$ and we can set $\sigma'(x) = \iota(x)$ if $\overline{up}(x) \notin X$.
- φ' is defined by $\varphi[x \leftarrow \sigma(x)]$[8]
- up' is defined by $\bigwedge_{x \in X \text{ and } \overline{up}(x) \notin X} \sigma'(x) := \overline{up}(x)$

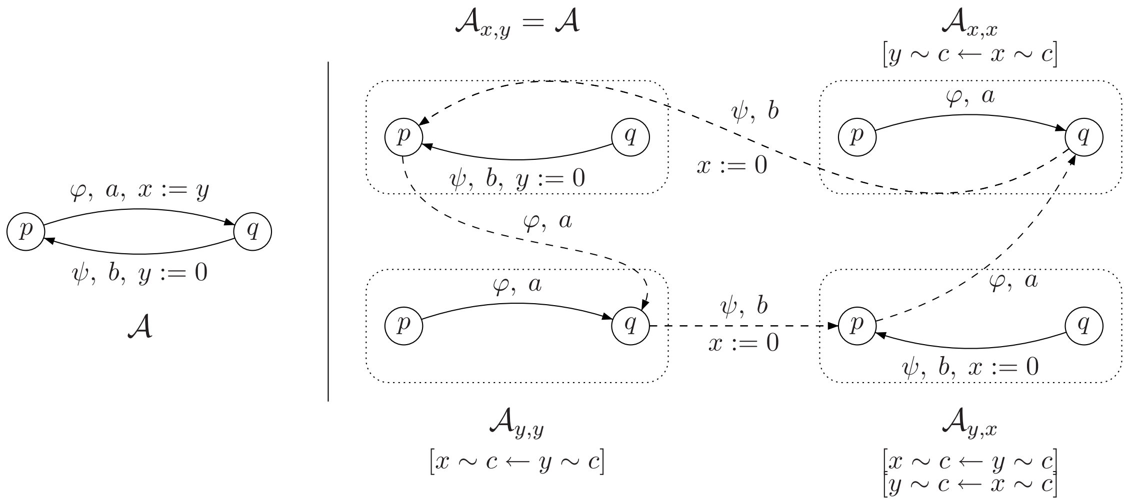
We define the relation $\mathcal{R}$ on $(Q \times \mathbb{T}^X) \times ((Q \times X^X) \times \mathbb{T}^X)$ by

$$\{(\langle q, \widehat{v}\rangle, \langle (q, \sigma), v\rangle) \mid q \in Q,\ \sigma \in X^X,\ v \in \mathbb{T}^X,\ \widehat{v} \in \mathbb{T}^X \text{ and } \widehat{v} = v \circ \sigma\}$$

The construction has been done precisely for $\mathcal{R}$ to be a bisimulation relation.
Note that the same construction can be done for timed automata having ε-transitions as well (in which case they are taken as normal actions) because automaton $\mathcal{B}$ does not have proper ε-transitions. $\square$

We illustrate the previous construction on the following example.

Example 10. Consider the automaton on the left of the figure below.



[8] The notation $\varphi[x \leftarrow \sigma(x)]$ is for the formula φ in which the variable x is replaced by $\sigma(x)$.

The construction described in the proof of the previous theorem applies to $\mathcal{A}$ and leads to the automaton drawn in the figure above, on the right (which consists of four copies of the original automaton, one for each application from the set $\{x, y\}$ onto the set $\{x, y\}$). In the copy $\mathcal{A}_{h_1,h_2}$ of $\mathcal{A}$, the value of x is stored in the clock h_1 whereas the value of y is stored in the clock h_2. A constraint $x \sim c$ must thus to be replaced by a constraint $h_1 \sim c$, as indicated on the figure. To illustrate the use of the ι injection: in state q of automaton $\mathcal{A}_{y,y}$, y has to be reset to zero, but y is the reference for clock x ($\sigma(x) = y$), we thus need to store the new value of y in a clock which plays no role, thus in x. In this case, $\iota(y) = x$, and thus $\sigma'(x) = y$ and $\sigma'(y) = x$. That's why the transition goes to state p of automaton $\mathcal{A}_{y,x}$. These two automata are strongly bisimilar.

We now pursue the study of updatable timed automata with deterministic updates by looking at the case wheresimple updates are of the form $x := d$.

Theorem 9. *Let $\mathcal{A} \in$ **Uta**$(\mathcal{C}(X),\mathcal{U}_{cst}(X))$ (resp. $\mathcal{A} \in$ **Uta**$_\varepsilon(\mathcal{C}(X),\mathcal{U}_{cst}(X))$). There exists a timed automaton $\mathcal{B} \in$ **Uta**$(\mathcal{C}(X),\mathcal{U}_0(X))$ (resp. $\mathcal{B} \in$ **Uta**$_\varepsilon(\mathcal{C}(X),\mathcal{U}_0(X))$) such that $\mathcal{A} \equiv_s \mathcal{B}$.*

Proof. Let $\mathcal{A}$ be a timed automaton in ***Uta***$(\mathcal{C}(X),\mathcal{U}_{\text{cst}}(X))$. Recall that from lemma 6, we assume without loss of generality that any update of $\mathcal{U}$ is in fact of the form $\{x := d \mid x \in X \text{ and } d \in \mathbb{Z}\}$.

We construct an automaton $\mathcal{B}$ in ***Uta***$(\mathcal{C}(X),\mathcal{U}_0(X))$, strongly bisimilar to $\mathcal{A}$. For every tuple $\alpha = (\alpha_x)_{x\in X}$ in $\mathbb{Z}^X$ such that for every clock x, $x := \alpha_x$ is a clock constraint appearing in $\mathcal{A}$, we construct a copy of the automaton $\mathcal{A}$, that we denote by $\mathcal{A}_\alpha$. Intuitively, in the automaton $\mathcal{A}_\alpha$, the value of the clock x is what the value should be in $\mathcal{A}$ decremented by α_x (α corresponds to a shift of the clocks, comparing with what their values should be in the initial automaton).
If $q \xrightarrow{\varphi,a,up} q'$ is a transition of $\mathcal{A}$, for every α, there will be a transition $q_\alpha \xrightarrow{\varphi_\alpha,a,up_\alpha} q'_{\alpha'}$ where:

- $\varphi_\alpha = \varphi[x \leftarrow x + \alpha_x]$,
- $up_\alpha = up[x := 0 \text{ instead of } x := c]$,
- $\alpha'_x = c$ if $x := c$ is part of the update up, $\alpha'_x = \alpha_x$ otherwise.

There are finitely many tuples $\alpha = (\alpha_x)_{x\in X}$, we thus only build finitely many copies of the initial automaton. We denote by $\mathcal{B}$ the union of all these automata $\mathcal{A}_\alpha$. The automaton $\mathcal{B}$ is obviously in ***Uta***$(\mathcal{C}(X),\mathcal{U}_0(X))$.

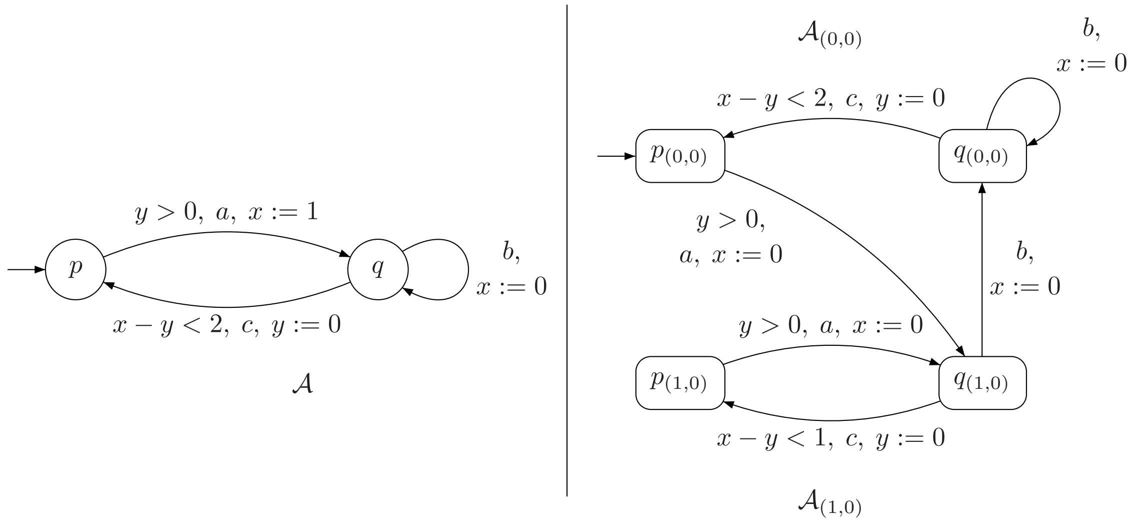
We define the relation $\mathcal{R}$ between the states of the transition system associated with $\mathcal{A}$ and the states of the transition system associated with $\mathcal{B}$ as:

$$(q, v)\mathcal{R}(q_\alpha, v_\alpha) \iff v = v_\alpha + \alpha$$

The relation $\mathcal{R}$ is trivially a bisimulation relation, which concludes the proof.
Like above, automaton $\mathcal{B}$ has no proper ε-transition, hence the same construction also holds for automata in ***Uta***$_\varepsilon(\mathcal{C}(X),\mathcal{U}_{\text{cst}}(X))$. □

We now illustrate the construction of the proof on the following example.

Example 11. Let us consider the automaton $\mathcal{A}$ drawn below, on the left. The previous construction gives the automaton on the right: here, we only need two copies of the automaton because the maximal constant for x is 1 whereas the maximal constant for y is 0.



If we consider now an updatable timed automaton which uses both updates of the forms $x := y$ and $x := d$, we can apply first the construction described in the proof of Theorem 8 and then the construction described in the proof of Theorem 9 to get a bisimilar classical timed automaton. We thus get the following result.

Corollary 2. *Let $\mathcal{C} \subseteq \mathcal{C}(X)$ be a set of clock constraints, and let*

$$\mathcal{U} \subseteq \textbf{\textit{Lu}}\left(\{x := d \mid x \in X \text{ and } d \in \mathbb{Q}\} \cup \{x := y \mid x,\ y \in X\}\right)$$

*Let $\mathcal{A} \in$ **Uta**$(\mathcal{C},\mathcal{U})$ (resp. $\mathcal{A} \in$ **Uta**$_\varepsilon(\mathcal{C},\mathcal{U})$). There exists a timed automaton $\mathcal{B} \in$ **Uta**$(\mathcal{C}(X),\mathcal{U}_0(X))$ (resp. $\mathcal{B} \in$ **Uta**$_\varepsilon(\mathcal{C}(X),\mathcal{U}_0(X))$) such that $\mathcal{A} \equiv_s \mathcal{B}$.*

We now consider the whole set of deterministic udpates and we will generalize the previous results. From the decidability results of section 5, we know that for general updatable timed automata, deterministic updates of the form $x := y + c$ can not always be replaced by resets. We thus need to restrict ourselves to diagonal-free timed automata with particular classes of updates. Note that the proof of the next theorem is much more involved than the proofs of the two previous theorems and that its results can not be considered any more as **"folfklore"**.

Recall that the system ($\mathcal{S}_{df}$) of linear inequations associated with a set of constraints and a set of updates has been defined at the end of section 5.2, page 19.

Theorem 10. *Let $\mathcal{C} \subseteq \mathcal{C}_{df}(X)$ be a set of diagonal-free clock constraints and*

$$\mathcal{U} \subseteq \textbf{\textit{Lu}}\left(\{x := d \mid x \in X \text{ and } d \in \mathbb{N}\} \cup \{x := y + d \mid x,\ y \in X \text{ and } d \in \mathbb{Z}\}\right)$$

*a set of deterministic updates such that the system ($\mathcal{S}_{df}$) of linear inequations associated with $\mathcal{C}$ and $\mathcal{U}$ has at least a solution. Let $\mathcal{A} \in$ **Uta**$(\mathcal{C},\mathcal{U})$ (resp. $\mathcal{A} \in$ **Uta**$_\varepsilon(\mathcal{C},\mathcal{U})$). There exists an automaton $\mathcal{B} \in$ **Uta**$(\mathcal{C}_{df}(X),\mathcal{U}_0(X))$ (resp. $\mathcal{B} \in$ **Uta**$_\varepsilon(\mathcal{C}_{df}(X),\mathcal{U}_0(X))$) such that $\mathcal{A} \equiv_s \mathcal{B}$.*

Proof. Let $\mathcal{A}$ be a timed automaton in **Uta**$(\mathcal{C},\mathcal{U})$. We build a timed automaton $\mathcal{B}$ in **Uta**$(\mathcal{C}(X),\mathcal{U}')$ where $\mathcal{U}' \subseteq \textbf{\textit{Lu}}\left(\{x := d \mid x \in X \text{ and } d \in \mathbb{N}\} \cup \{x := y \mid x,\ y \in X\}\right)$ which will be strongly bisimilar to $\mathcal{A}$. Applying Corollary 2 will give the proof.

We consider integer constants $(\max_x)_{x \in X}$, solutions of the system ($\mathcal{S}_{df}$) (see page 19) for the automaton $\mathcal{A}$. For every $\alpha = (\alpha_x)_{x \in X} \in \mathbb{Z}^X$ such that for every clock x, $\alpha_x \leq \max_x + 1$, for every state q of $\mathcal{A}$, we consider a copy q_α of q. Intuitively, in the state q_α, the value of the clock x will be the value this clock should have in q, minus α_x (α can be seen as a shift of the clocks w.r.t. their values in the initial automaton).

If $q \xrightarrow{\varphi,a,up} q'$ is a transition of $\mathcal{A}$, we add a transition $q_\alpha \xrightarrow{\varphi_\alpha,a,up_\alpha} q'_{\alpha'}$, for every α with:

- $\varphi_\alpha = \varphi[x \leftarrow x + \alpha_x]$,
- $up_\alpha = up[x := y$ instead of $x := y + c]$,
- $\alpha'_x = \begin{cases} \alpha_y + c \text{ if } x := y + c \text{ update of } up \\ 0 \qquad\;\; \text{if } x := c \text{ update of } up \end{cases}$

  If the value of α'_x computed in this way satisfies that $\alpha'_x > \max_x$, then we update α'_x to $\max_x + 1$.

  We say that α' is obtained from α **in an elementary way** thanks to the update up.

The number of tuples $\alpha = (\alpha_x)_{x \in X} \in \mathbb{Z}^X$ such that for every clock x, $\alpha_x \leq \max_x + 1$ is infinite. We did thus construct, for every state q, an infinite number of copies. However, we will prove that, from the initial states indexed by $(0, \ldots, 0)$, only a finite number of such states are reachable.

It is of course sufficient to prove that the set of tuples α such that a state q_α is reachable, is lower bounded. Assume that it is not the case. There exists a sequence of tuples $(\alpha^{(i)})_{i \geq 0}$ such that $\alpha^{(0)} = (0, \ldots, 0)$, and for every i, $\alpha^{(i+1)}$ is obtained from $\alpha^{(i)}$ in an elementary way thanks to an update up_i, and moreover, the sequence $(\alpha_x^{(i)})_{i \geq 0}$ tends to $-\infty$ (for a given clock x). By definition of $\mathcal{U}$, every up_i can be written in the form:

$$\bigwedge_{x \in X_1} x := d_x \wedge \bigwedge_{\substack{x \in X_2 \\ c_x < 0}} x := y_x + c_x \wedge \bigwedge_{\substack{x \in X_3 \\ c_x \geq 0}} x := y_x + c_x$$

with X_1, X_2 and X_3 disjoint sets. We thus set

$$up'_i := \bigwedge_{\substack{x \in X_2 \\ c_x < 0}} x := y_x + c_x$$

and we define the sequence $(\beta^{(i)})_{i \geq 0}$ with:

$$\begin{cases} \beta^{(0)} = \alpha^{(0)} \\ \beta^{(i+1)} \text{ is obtained in an elementary way from } \beta^{(i)} \text{ thanks to } up'_i \end{cases}$$

It is easy to verify that the sequence $(\beta^{(i)})_{i \geq 0}$ is decreasing, and non-stationary (for the natural order on the tuples of integers) because $(\alpha_x^{(i)})_{i \geq 0}$ tends to $-\infty$ for some clock x.

Let z_1 be a clock such that the sequence $(\beta_{z_1}^{(i)})_{i \geq 0}$ tends to $-\infty$. There exists at least an update of the form $z_1 := z_2 + c_1$ belonging to $\mathcal{U}$ (thus with $c_1 < 0$) such that the sequence $(\beta_{z_2}^{(i)})_{i \geq 0}$ also tends to $-\infty$. In this way, we can construct a sequence of clocks $(z_p)_{p \geq 1}$ such that:

- there exists an update $z_p := z_{p+1} + c_p$ in $\mathcal{U}$ (with $c_p < 0$),
- for every $p \geq 1$, the sequence $(\beta_{z_p}^{(i)})_{i \geq 0}$ tends to $-\infty$.

The set of clocks is finite, there exists thus $p < q$ such that $z_p = z_q$. However, the constants $(\max_x)_{x \in X}$ are solutions of the system ($\mathcal{S}_{df}$), page 19 and this system contains in particular the inequations

$$\begin{aligned} \max_{z_p} &\leq \max_{z_{p+1}} + c_p \quad \text{with } c_p < 0 \\ &\vdots \\ \max_{z_{q-1}} &\leq \max_{z_q} + c_{q-1} \quad \text{with } c_{q-1} < 0 \end{aligned}$$

In particular the constant $\max_{z_p} = \max_{z_q}$ has to satisfy $\max_{z_p} < \max_{z_p}$, which is not possible.

Thus we have proven that the set of states q_α which are reachable is finite. We denote by $\mathcal{B}$ the automaton we just constructed. This automaton belongs to ***Uta***$(\mathcal{C}(X), \mathcal{U}')$.

We define the relation $\mathcal{R}$ as follows, between the states of the transition system associated with $\mathcal{A}$, and the states of the transition system associated with $\mathcal{B}$:

$$(q, v)\mathcal{R}(q_\alpha, v_\alpha) \iff \begin{cases} v \text{ and } v_\alpha + \alpha \text{ are equivalent for the region equivalence } \mathcal{R}_{(\max_x)_{x \in X}} \\ v(x) \leq \max_x \implies v(x) = v_\alpha(x) + \alpha_x \text{ for every } x \in X \end{cases}$$

We will prove that $\mathcal{R}$ is a bisimulation relation.

Let us assume that $(q, v)\mathcal{R}(q_\alpha, v_\alpha)$ and that $(q, v) \xrightarrow{a} (q', v')$. It means that there exists a transition $q \xrightarrow{\varphi,a,up} q'$ in $\mathcal{A}$ such that $v \models \varphi$ and $v' = up(v)$. In $\mathcal{B}$, there is a transition $q_\alpha \xrightarrow{\varphi_\alpha,a,up_\alpha} q'_{\alpha'}$. We set $v'_{\alpha'} = up_\alpha(v_\alpha)$ and we will prove that $(q', v')\mathcal{R}(q'_{\alpha'}, v')$.

- if x is a clock such that $x := c$ belongs to up, then $x := c$ also belongs to up_α.
  Thus, $v'_{\alpha'}(x) = c = v'(x)$ and $\alpha'_x = 0$.
- if x is a clock such that $x := y + c$ belongs to up, then $x := y$ also belongs to up_α,
  - Assume that $v'(x) \in I_x$ with $I_x \leq \max_x$ (*i.e.* that $I_x =]d-1; d[$ or $[d]$ with $d \leq \max_x$).
    We want to show that $v'(x) = v'_{\alpha'}(x) + \alpha'_x$. To this aim, we compute

    $$v'_{\alpha'}(x) + \alpha'_x = v_\alpha(y) + \alpha'_x \text{ because } x := y \text{ belongs to } up_\alpha$$

    We distinguish two cases:
    1. If $\alpha'_x \leq \max_x$, we then get that

       $$v'_{\alpha'}(x) + \alpha'_x = v_\alpha(y) + \alpha_y + c$$

       However, we have that $(q, v)\mathcal{R}(q_\alpha, v_\alpha)$ and $v(y) \leq \max_y$ (because $v'(x) = v(y)+c \leq \max_x$ and $\max_x \leq \max_y + c$), thus

       $$v'_{\alpha'}(x) + \alpha'_x = v(y) + c = v'(x)$$

    2. If $\alpha'_x > \max_x$, it means that $\alpha_y + c > \max_x$. However,

       $$v'(x) = v(y) + c = v_\alpha(y) + \alpha_y + c > \max_x$$

       It is of course not possible because we did assume that $v'(x) \leq \max_x$.
  - Assume that $v'(x) > \max_x$. We distinguish two cases:
    1. If $\alpha'_x > \max_x$, then $v'_{\alpha'}(x) + \alpha'_x > \max_x$.
    2. If $\alpha'_x \leq \max_x$, then $v'_{\alpha'}(x) + \alpha'_x v_\alpha(y) + \alpha_y + c$. There are also two cases:
       (i) if $v_\alpha(y) + \alpha_y \leq \max_y$, then

       $$v'_{\alpha'}(x) + \alpha'_x = v(y) + c = v'(x) > \max_x$$

       (ii) if $v_\alpha(y) + \alpha_y > \max_y$, then as $\max_x \leq \max_y + c$, we get that $v'_{\alpha'}(x) + \alpha'_x > \max_x$.

    In all cases, we have seen that $v'_{\alpha'}(x) + \alpha'_x > \max_x$, and that is precisely what we wanted.
  - the change between up and up_α keeps the relative order of the fractional parts.

We thus get that $(q', v')\mathcal{R}(q'_{\alpha'}, v'_{\alpha'})$. The reverse is very similar.

We did thus exhibit a bisimulation relation between $\mathcal{A}$ and $\mathcal{B}$. □

Remark 8. Up to the (un)decidability results (*cf* section 4), we cannot extend the previous result to timed automata that also use diagonal clock constraints, because this leads to an undecidable model. It is interesting to understand why the previous proof cannot be extended and thus where the diagonal-free hypothesis is fundamental. In order to have a finite number of copies of each state, we set the value $\max_x +1$ to α'_x whenever the computed value is greater than $\max_x +1$. This change does not disturb the truth or the falsity of diagonal-free clock constraints, but can change the truth or the falsity of diagonal clock constraints.

Example 12. In this case also, we consider a simple example. The two automata drawn on figure 2 are strongly bisimilar. The one on the right results from the construction described above, taking as initial automaton the one on the left. The maximal constants are $\max_x = 0$ and $\max_y = 1$.

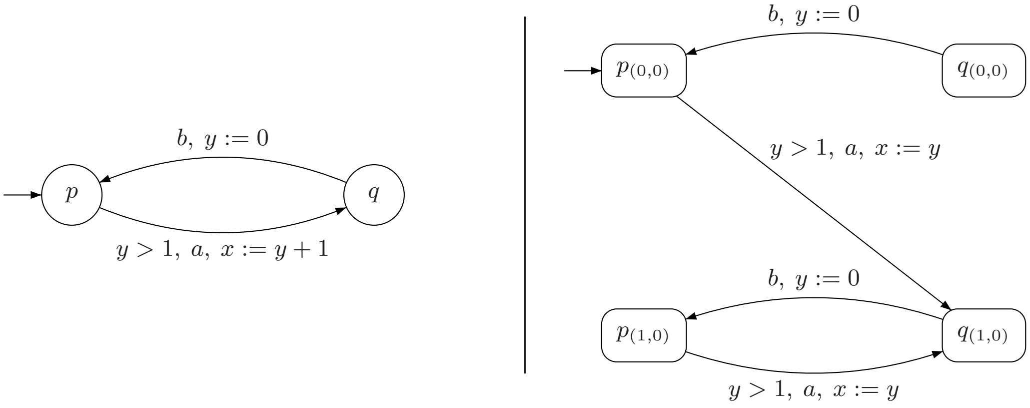


Fig. 2. Two strongly bisimilar automata

6.4 Expressiveness of Non-Deterministic Updates

We now study the general case of non-deterministic updates. From the example of section 6.2, it is false to say that any updatable timed automaton with non-deterministic updates is strongly equivalent to a classical timed automaton. We will thus concentrate our efforts on weak similarity. We will prove that any updatable timed automaton with non-deterministic updates, from a decidable class, is weakly bisimilar to a timed automaton with ε-transitions. But, as it will appear, the constructions are much more technical than in the case of deterministic updates. We first deal with diagonal-free automata.

Construction for diagonal-free clock constraints. We propose a normal form for diagonal-free updatable timed automata. Let $(\max_x)_{x\in X}$ be a family of integer constants. In what follows we only consider clock constraints $x \sim c$ with $c \leq \max_x$. As defined in section 5.2, we set:

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

A clock constraint φ is said to be *total* if φ is a conjunction $\bigwedge_{x\in X}(x \in I_x)$ where for each clock x, I_x is an element of $\mathcal{I}_x$. Any diagonal-free clock constraint bounded by the constants $(\max_x)_{x\in X}$ is equivalent to a disjunction of total clock constraints.

We also define

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

An update up_x is said *elementary* if it is of one of the following forms:

- $x :\in I_x$ with $I_x \in \mathcal{I}_x$,
- $x := y + c \wedge x :\in I'_x$ with $I'_x \in \mathcal{I}'_x$ and $\max_x \leq \max_y + c$,
- $\left(\bigwedge_{y\in H} x :< y + c \wedge x :\in I'_x\right)$ with $H \subseteq X$, $I'_x \in \mathcal{I}'_x$ and $\forall y \in H$, $\max_x \leq \max_y + c$,
- $\left(\bigwedge_{y\in H} x :> y + c \wedge x :\in I'_x\right)$ with $H \subseteq X$, $I'_x \in \mathcal{I}'_x$ and $\forall y \in H$, $\max_x \leq \max_y + c$.

An elementary update up_x is *compatible* with a total constraint $\bigwedge_{x\in X}(x \in I_x)$ if:

- $I_y + c \subseteq I'_x$ whenever up_x is $x := y + c \wedge x :\in I'_x$,
- for any $y \in H$, $I_y + c \subseteq I'_x$ whenever up_x is $((\bigwedge_{y\in H} x :\sim y + c) \wedge x :\in I'_x)$ and $I'_x = I_x$.

Definition 5. *Let $(\max_x)_{x\in X}$ be integer constants and let $\mathcal{A}$ be a timed automaton in* **Uta**$(\mathcal{C}_{df}(X), \mathcal{U}(X))$. *We say that $\mathcal{A}$ is in* normal form *for the constants $(\max_x)_{x\in X}$ whenever for every transition $q \xrightarrow{\varphi,a,up} q'$ of $\mathcal{A}$, the following holds:*

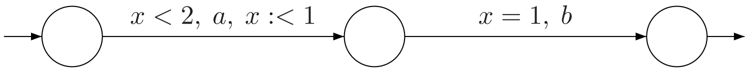
– φ *is a total clock constraint,*
– $up = \bigwedge_{x \in X} up_x$ *where for every clock* x, up_x *is an elementary update, compatible with* φ.

Applying classical rules of propositional calculus and splitting the transitions, we easily obtain the normal form for diagonal-free updatable timed automata (recall that we restrict here our work to updates defined by ($\diamondsuit_{df}$), page 19):

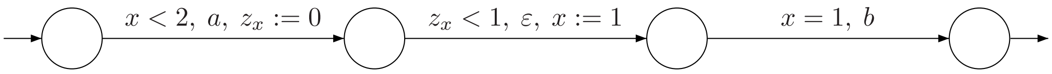
Proposition 10. *Let* $\mathcal{C}$ *be a set of diagonal-free clock constraints and* $\mathcal{U}$ *be a set of updates defined by the grammar (*$\diamondsuit_{df}$*). We assume that the system (*$\mathcal{S}_{df}$*) has a solution,* $(\max_x)_{x \in X}$*. Any timed automaton of* ***Uta***$(\mathcal{C},\mathcal{U})$ *is strongly bisimilar to a timed automaton of* ***Uta***$(\mathcal{C}_{df}(X),\mathcal{U}(X))$ *which is in normal form for the constants* $(\max_x)_{x \in X}$.

Before stating our main result about the expressiveness of diagonal-free updatable timed automata, let us try to illustrate the difficulties and the techniques that we will use on two toy examples.

Example 13. Consider the following automaton:

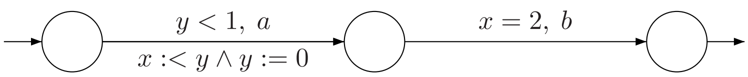


The timed language recognized by this automaton is $\{(a,t)(b,t') \mid 0 \leq t < 2 \text{ and } 0 < t' - t < 1\}$.
The previous automaton can be weakly simulated by the following automaton, which only has deterministic updates:

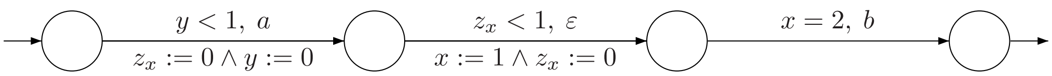


The non-deterministic update of the first automaton has been replaced by a silent action. The clock z_x which has been added represents the fractional part of x and thus checks whether it does not become bigger than 1.

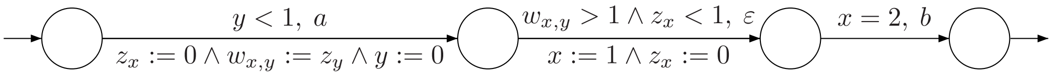
Example 14. Let us consider the following automaton:



The timed language recognized by this automaton is $\{(a,t)(b,t') \mid t < 1 \text{ and } t' > 2\}$.
A first (wrong) idea is to perform the transformation above:



However the transformation is not correct. This automaton accepts for example the timed word $(a, 0.5)(b, 1.8)$, which is not recognized by the initial automaton.
To avoid this problem, we can add a new clock, $w_{x,y}$ which aims at keeping in mind that, when x has been updated, the value of x was less than the value of y. This leeds to the following automaton:



When the second transition is taken, the value of x is set to 1 (this transition is chosen at a non-determinisc date), and to ensure that the value of y was greater than x, we add the constraint $w_{x,y} > 1$. The clock $w_{x,y}$ thus stores the value of y when an update $x :< y$ is done in the original automaton. Clock y can then be reset safely, information on the old value of x and thus on the difference $x - y$ is stored in $w_{x,y}$. It is easy to verify that this automaton recognizes the same timed language as the initial automaton.

We will generalize the constructions of these two examples to prove the next theorem on the expressiveness of updatable automata with non-deterministic updates and diagonal-free clock constraints.

Theorem 11. *Let $\mathcal{C}$ be a set of diagonal-free clock constraints and $\mathcal{U}$ be a set of updates defined by the grammar ($\diamondsuit_{df}$). We assume in addition that the system ($\mathcal{S}_{df}$) has a solution for $\mathcal{C}$ and $\mathcal{U}$. Let $\mathcal{A} \in$ **Uta**$(\mathcal{C},\mathcal{U})$ (resp. $\mathcal{A} \in$ **Uta**$_\varepsilon(\mathcal{C},\mathcal{U})$). There exists an automaton $\mathcal{B} \in$ **Uta**$_\varepsilon(\mathcal{C}_{df}(X),\mathcal{U}_0(X))$ such that $\mathcal{B} \succcurlyeq_w \mathcal{A}$ and $\mathcal{A} \equiv_\ell \mathcal{B}$.*

Proof. Thanks to lemma 6 and proposition 10, we assume that all constants appearing in $\mathcal{A}$ are integers and that $\mathcal{A}$ is in normal form for some constants $(\max_x)_{x \in X}$.
A clock x is said *fixed* if the last update for x was either of the form $x := c$ or $(x := y + c \wedge x :\in I'_x)$ where the clock y was itself fixed. A clock which is not fixed is said *floating*. The terminology "floating" comes from the fact that the value of a floating clock is not precisely known, we only know the interval of the form $]d; d+1[$ to which it belongs.
The transformation algorithm constructs (a lot of) copies of the original automaton $\mathcal{A}$, by adding suitable clocks, transforming the transitions and adding silent actions in order to go from one copy to another.

Adding clocks.
For any clock x in X, we define a clock z_x which intuitively represents the fractional part of x.
For any pair of clocks (x, y), we also define two clocks, $w_{x,y}$ and $w'_{x,y}$, which will compare the fractional parts of x and y. Let $\overline{X}$ be the set of these $2|X|^2$ additional clocks. We will explain their precise roles along the construction.

Duplication of the original automaton.
Let us consider a subset Y of X, that corresponds intuitively to the floating clocks, and a partial order $\prec$ defined on Y, which represents the relative order of the fractional parts of the clocks in Y.
Moreover, for any clock y of Y, we define an interval I_y, of the form $]d; d+1[$ with $0 \le d < \max_y$. The clock y will be supposed to be in the interval I_y.
Finally, we consider a subset Z of $\overline{X}$, whose role will be explained below.

For any tuple $\tau = ((I_y)_{y \in Y}, \prec, Z)$, we construct a copy $\mathcal{A}_\tau$ of the automaton $\mathcal{A}$. On each transition of $\mathcal{A}_\tau$, we add the clock constraint

$$\bigwedge_{y \in Y} y \in I_y \wedge \bigwedge_{x \in X} z_x < 1$$

Some such constraints are trivially equivalent to "False", in which case the corresponding transition can be erased.
We denote by T the set of all the tuples τ described above.

Fixed clocks.
When the fractional part of a fixed clock reaches the value 1, we stay in the same copy of the automaton. To ensure this, in every copy $\mathcal{A}_\tau$ with $\tau = ((I_y)_{y \in Y}, \prec, Z)$, we add on each state and for every clock $x \in X \setminus Y$, a loop labelled by $(z_x = 1,\ \varepsilon,\ z_x := 0)$.

Floating clocks.

We can fix some floating clocks withs a silent action. Of course, a clock can be fixed only by reaching an integer value. Among the floating clocks, the first ones which will reach first the upper bound of their interval are those maximal for the preorder. Formally, let $\mathcal{A}_\tau$ with $\tau = ((I_y)_{y\in Y}, \prec, Z)$ and let M be the set of maximal elements for $\prec$. For any state q of $\mathcal{A}_\tau$, we construct an ε-transition leading to the copy of q in the automaton $\mathcal{A}_{\tau'}$ such that $\tau' = ((I_y)_{y\in Y'}, \prec', Z')$ where:

$$\begin{cases} Y' = Y \setminus M \\ \prec' = \prec \cap (Y' \times Y') \\ Z' = Z \setminus \{w_{x,y}, w'_{x,y} \mid x \in M\} \end{cases}$$

This ε-transition is labelled by the clock constraint

$$\bigwedge_{x\in M,\ w_{x,y}\in Z} (w_{x,y} \geq 1) \wedge \bigwedge_{x\in M,\ w'_{x,y}\in Z} (w'_{x,y} < 1) \wedge \bigwedge_{y\in Y} (z_y < 1)$$

and the update

$$\bigwedge_{y\in M} y := \mathsf{sup}(I_y)$$

where $\mathsf{sup}(I_y)$ represents the upper bound of I_y, *i.e.* $d+1$ if $I_y =]d; d+1[$.

The existence of a clock $w_{x,y}$ (resp. $w'_{x,y}$) shows that an update of the form $x :< y+c$ (resp. $x :> y+c$) has been used previously. The clock constraint $w_{x,y} \geq 1$ (resp. $w'_{x,y} < 1$) ensures that we did really simulate such an update.

Modification of the transitions.

We consider a copy $\mathcal{A}_\tau$ with $\tau = ((I_y)_{y\in Y}, \prec, Z)$ and a transition $(q_\tau, \varphi, a, up, q'_\tau)$ of this copy. This transition will be replaced by a transition $(q_\tau, \varphi, a, \widehat{up}, q'_{\widehat{\tau}})$ where $q'_{\widehat{\tau}}$ is the state, corresponding to q'_τ in an other copy $\mathcal{A}_{\widehat{\tau}}$ with $\widehat{\tau}(\widehat{I_y})_{y\in\widehat{Y}}, \widehat{\prec}, \widehat{Z})$ which will be made precise below.

The components $\widehat{Y}$, $(\widehat{I_y})_{y\in\widehat{Y}}$, $\widehat{\prec}$ and $\widehat{up}$ will be built inductively by considering one after the other the updates appearing in up.

The new update $\widehat{up}$ will only be defined thanks to deterministic updates (of the form $x := c$ or $x := y+c$). Initially, we set $\widehat{Y} = Y$, $\widehat{I_y} = I_y$ for every $y \in Y$, $\widehat{\prec} = \prec$, $\widehat{up} = \varnothing$ and $\widehat{Z} = Z$.

Before listing all the possible updates, we explain the role of the set Z, which has not been precised yet. Assume that the clock x has been updated thanks to $x :< y + c$ where y is a fixed clock. The clock x becomes floating. We use the clock z_x in order to store the fractional part of x, we reset this clock to zero. We also need to keep in mind the current value of the fractional part of y, stored until now "in" the clock z_y. As z_x must stay less than z_y, z_y *must* reach 1 before z_x does. Of course, if the clock y is not updated, this can be checked using the clock z_y, but if the clock y is also updated, or is updated before z_y reaches 1, the old value of z_y will be forgotten. We thus add the clock $w_{x,y}$ to the set Z and we set $w_{x,y} := z_y$. The clock $w_{x,y}$ will keep in mind the old value of z_y, whatever the clock y becomes. The property that we now need to check is that $w_{x,y} \geq 1$. The role of the clocks $w'_{x,y}$ is similar, but they are used for the updates of the form $x :> y + c$, where y is a fixed clock. The condition "z_x reaches the value 1 before z_y" is checked thanks to the clock constraint $w'_{x,y} < 1$. Example 14 illustrates the use of these clocks.

As said before, we now list all the possible values for the updates:

- if up_x is $x := c$, we just need to consider x as a fixed clock:

$$\widehat{Y} \leftarrow \widehat{Y} \setminus \{x\},\ \widehat{Z} \leftarrow \widehat{Z} \setminus \{w_{x,y}, w'_{x,y} \mid y \in X\},\ \widehat{up} \leftarrow \widehat{up} \wedge x := c \wedge z_x := 0$$

- if up_x is $x :\in I'_x$,

1. if $I'_x =]c_x; +\infty[$, then
   $\widehat{Y} \leftarrow \widehat{Y} \setminus \{x\}$, $\widehat{Z} \leftarrow \widehat{Z} \setminus \{w_{x,y}, w'_{x,y} \mid y \in X\}$, $\widehat{up} \leftarrow \widehat{up} \wedge x := c_x + 1 \wedge z_x := 0$
2. if $I'_x =]c; c+1[$, then
   $\widehat{Y} \leftarrow \widehat{Y} \cup \{x\}$, $\widehat{Z} \leftarrow \widehat{Z} \setminus \{w_{x,y}, w'_{x,y} \mid y \in X\}$, $\widehat{\precsim}$ is a total preorder compatible with $\widehat{\precsim}$ on the set $\widehat{Y} \setminus \{x\}$, $\widehat{up} \leftarrow \widehat{up} \wedge z_x := 0$

– if up_x is $x := y + c \wedge x :\in I'_x$,
  1. if $y \notin Y$,
     - $\widehat{Y} \leftarrow \widehat{Y} \setminus \{x\}$,
     - $\widehat{Z} \leftarrow \widehat{Z} \setminus \{w_{x,y}, w'_{x,y} \mid y \in X\}$
     - $\widehat{up} \leftarrow \widehat{up} \wedge x := y + c \wedge z_x := z_y$
  2. if $y \in Y$,
     - if I'_x is bounded,
       · $\widehat{Y} \leftarrow \widehat{Y} \cup \{x\}$,
       · $\widehat{Z} \leftarrow \widehat{Z} \setminus \{w_{x,y}, w'_{x,y} \mid y \in X\}$,
       · $x \widehat{\precsim} y$ and $y \widehat{\precsim} x$,
       · $\widehat{I}_x = I'_x$,
       · $\widehat{up} \leftarrow \widehat{up} \wedge z_x := x_y$
     - if I'_x is not bounded, *i.e.* $I'_x =]c_x; +\infty[$,
       · $\widehat{Y} \leftarrow \widehat{Y} \setminus \{x\}$,
       · $\widehat{Z} \leftarrow \widehat{Z} \setminus \{w_{x,y}, w'_{x,y} \mid y \in X\}$,
       · $\widehat{up} \leftarrow \widehat{up} \wedge x := c_x + 1 \wedge z_x := z_y$

- if up_x is $\left(\bigwedge_{y \in H} x :< y + c\right) \wedge x :\in I'_x$, we set $H_1 = H \cap Y$ and $H_2 = H \setminus Y$ and
  - if I'_x is bounded,
    · $\widehat{Y} = \widehat{Y} \cup \{x\}$,
    · $\widehat{Z} = (\widehat{Z} \setminus \{w_{x,y}, w'_{x,y} \mid y \in X\}) \cup \{w_{x,y} \mid y \in H_2\}$,
    · $x \widehat{\precsim} y$ and $y \widehat{\not\precsim} x$ if $y \in H_1$,
    · $\widehat{up} \leftarrow \widehat{up} \wedge z_x := 0 \wedge \bigwedge_{y \in H_2} w_{x,y} := z_y$.
  - if I'_x is $]c_x; +\infty[$,
    · $\widehat{Y} = \widehat{Y} \setminus \{x\}$,
    · $\widehat{Z} = (\widehat{Z} \setminus \{w_{x,y}, w'_{x,y} \mid y \in X\})$,
    · $\widehat{up} \leftarrow \widehat{up} \wedge x := c_x + 1 \wedge z_x := 0$.
- if up_x is $\left(\bigwedge_{y \in H} x :> y + c\right) \wedge x :\in I'_x$, we set $H_1 = H \cap Y$ and $H_2 = H \setminus Y$ and
  - if I'_x is bounded,
    · $\widehat{Y} = \widehat{Y} \cup \{x\}$,
    · $\widehat{Z} = (\widehat{Z} \setminus \{w_{x,y}, w'_{x,y} \mid y \in X\}) \cup \{w'_{x,y} \mid y \in H_2\}$,
    · $y \widehat{\precsim} x$ and $x \widehat{\not\precsim} y$ if $y \in H_1$,

· $\widehat{up} \leftarrow \widehat{up} \wedge z_x := 0 \wedge \bigwedge_{y \in H_2} w'_{x,y} := z_y$.

- if I'_x is $]c_x; +\infty[$,
  · $\widehat{Y} = \widehat{Y} \setminus \{x\}$,
  · $\widehat{Z} = (\widehat{Z} \setminus \{w_{x,y}, w'_{x,y} \mid y \in X\})$,
  · $\widehat{up} \leftarrow \widehat{up} \wedge x := c_x + 1 \wedge z_x := 0$.

It remains to prove that the resulting automaton weakly simulates the initial automaton and that, in addition, it recognizes the same timed language.

We now define a relation $\mathcal{R}$, which will be a simulation relation. Roughly, a state of the original automaton will be in relation with all the copies of this state in the copies of the automaton. The set of states of the timed transition system associated with $\mathcal{A}$ is $Q \times \mathbb{T}^X$, whereas the set of states of the transition system associated with $\mathcal{B}$ is:

$$\{q_\tau \mid q \in Q \text{ and } \tau \in T\} \times \mathbb{T}^{X \cup \{z_x \mid x \in X\} \cup Z}$$

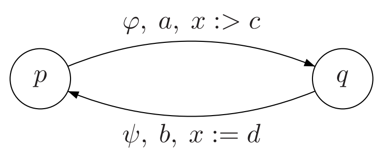
We define the relation $\succcurlyeq$ by

$$\succcurlyeq \; = \; \left\{ ((q_\tau, v'), (q, v)) \;\middle|\; \begin{array}{l} \forall y \in Y,\ v(y) \in I_y \text{ and } 0 \leq v'(z_y) \leq 1, \\ \forall y \in X \setminus Y,\ \text{either } v(y) = v'(y) \text{ or } (v(y) > c_y \text{ and } v'(y) > c_y), \\ y_1 \prec y_2 \implies \mathsf{frac}(v(y_1)) \leq \mathsf{frac}(v(y_2)), \\ w_{x,y} \in Z \implies \mathsf{frac}(v(x)) < v'(w_{x,y}) \\ \quad \text{and } w'_{x,y} \in Z \implies \mathsf{frac}(v(x)) > v'(w'_{x,y}). \end{array} \right\}$$

It is easy but tiresome to prove that $\succcurlyeq$ is a simulation relation and that the automaton $\mathcal{B}$ recognizes the same timed language as the initial automaton.

The automaton which has been constructed only has deterministic updates and diagonal-free clock constraints. We finally use Corollary 2 to conclude the proof of theorem 11. $\square$

Example 15. Consider the timed automaton below:



The transformation of the proof builds the automaton depicted on figure 3 (in this case, no clock $w_{x,y}$ or $w'_{x,y}$ is needed). This construction suffers from an important combinatorics explosion, we thus only draw a small part of the resulting automaton, it should be sufficient for understanding the construction.

Let us describe this automaton. There is only one clock x. One copy for each interval $]\alpha; \alpha+1[$ (with $c < \alpha \leq \max_x$) is needed. The transition going up on the right of the figure represents the fact that clock x has reached the upper bound of interval $]\alpha; \alpha+1[$ where it was floating. This transition can be taken in a non-deterministic way, it thus fix *a posteriori* the value clock x had after the update $x :> c$. Loops on the upper automaton represent when the value for x through the update $x :> c$ is taken as an integer value or a value greater than the maximal constant (in which case, the precise value is not important, we just need to know that it is bigger than $\max_x$, thus we set it arbitrarly to $\max_x + 1$.

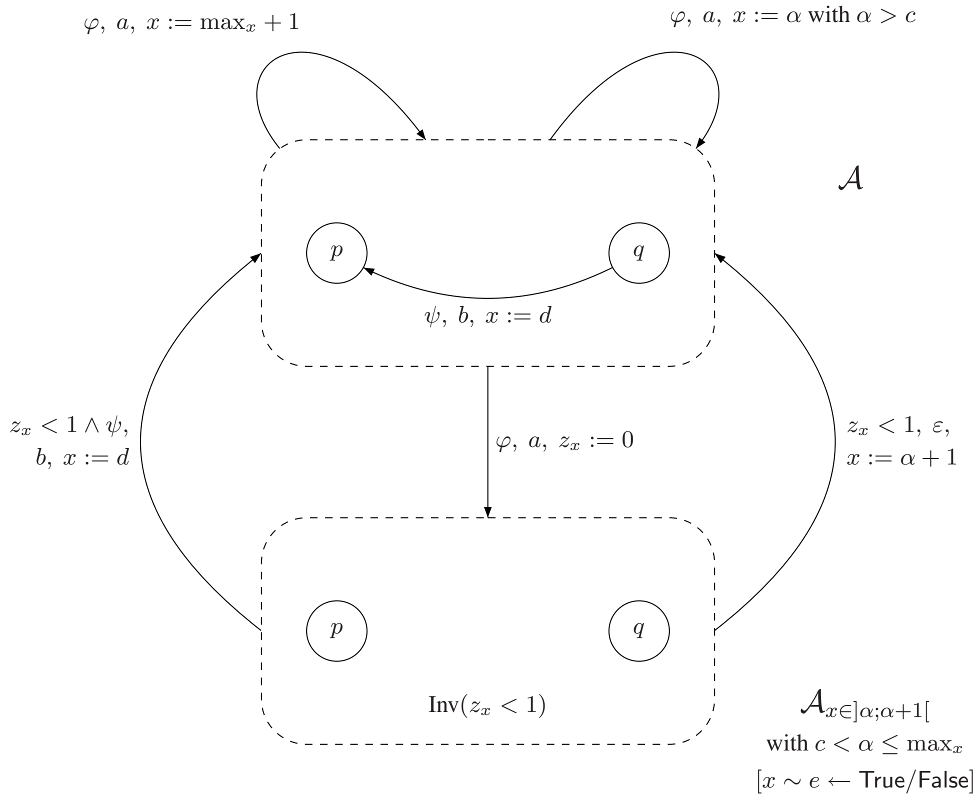


Fig. 3. Removing the non-deterministic updates

Construction for general clock constraints. We consider now updatable timed automata with general clock constraints. As in the previous section, we define a normal form for these automata. We consider again the sets $\mathcal{I}_x$, $\mathcal{I}'_x$, $\mathcal{J}_{x,y}$ defined in sections 5.2 and 5.3. We will say that a clock constraint

$$\bigwedge_{x \in X} x \in I_x \wedge \bigwedge_{x,y \in X} x - y \in J_{x,y}$$

is *total* whenever for every clock x, $I_x \in \mathcal{I}_x$ and for all clocks $x, y \in X$, $J_{x,y} \in \mathcal{J}_{x,y}$. We will say that an update up_x for the clock x is *strictly elementary* whenever it is of one of the following forms:

- $x := c$ with $0 \leq c \leq \max_x$,
- $x :\in I''_x$ with $I''_x \in \mathcal{I}''_x$ ($\mathcal{I}''_x$ is the set $\{]c; c+1[\mid 0 \leq c < \max_x\}$),
- $(x := y \wedge x :\in I'_x)$ with $I'_x \in \mathcal{I}'_x$.

A strictly elementary update up_x is *compatible* with a total clock constraint

$$\bigwedge_{x \in X} x \in I_x \wedge \bigwedge_{x,y \in X} x - y \in J_{x,y}$$

if $I_y \subseteq I'_x$ as soon as up_x is $x := y \wedge x :\in I'_x$.

Definition 6. *Let* $((\max_x)_{x \in X}, (\max_{x,y})_{x,y \in X})$ *be a tuple of constants and let* $\mathcal{A}$ *be a timed automaton in* ***Uta***$(\mathcal{C}(X), \mathcal{U}(X))$. $\mathcal{A}$ *is said to be in* normal form *for the constants* $((\max_x)_{x \in X}, (\max_{x,y})_{x,y \in X})$ *if for every transition* $q \xrightarrow{\varphi, a, up} q'$ *of* $\mathcal{A}$*:*

– *φ is a total clock constraint, and*
– *$up = \bigwedge_{x \in X} up_x$ with for every clock x, up_x is a strictly elementary update, compatible with φ.*

Applying the classical rules of the propositional calculus and splitting the transitions, we obtain the normal form for the timed automata with general clock constraints (recall that updates are restricted to the definition $\diamondsuit_{gen}$, page 24).

Proposition 11. *Let $\mathcal{C}$ be a set of general clock constraints and let $\mathcal{U}$ be a set of updates generated by the grammar ($\diamondsuit_{gen}$). We assume that the system ($\mathcal{S}_{gen}$) has a solution, $((\max_x)_{x \in X}, (\max_{x,y})_{x,y \in X})$. Every automaton in **Uta**$(\mathcal{C},\mathcal{U})$ is strongly bisimilar to an automaton in **Uta**$(\mathcal{C}(X),\mathcal{U}(X))$ which is in normal form for the constants $((\max_x)_{x \in X}, (\max_{x,y})_{x,y \in X})$.*

When we are interested in decidable subclasses of timed automata with general clock constraints, we must restrict the set of updates which we consider. As will be established in the following theorem, the decidable timed automata can be weakly simulated by classical timed automata with silent actions.

Theorem 12. *Let $\mathcal{C}$ be a set of general clock constraints and $\mathcal{U}$ be a set of updates generated by the grammar ($\diamondsuit_{gen}$). Let $\mathcal{A}$ be an automaton in **Uta**$(\mathcal{C},\mathcal{U})$. There exists an automaton $\mathcal{B}$ in **Uta**$_\varepsilon(\mathcal{C}(X),\mathcal{U}_0(X))$ such that $\mathcal{B} \succcurlyeq_w \mathcal{A}$ and $\mathcal{A} \equiv_\ell \mathcal{B}$.*

The proof is similar to the one of theorem 11, and is even simpler since we do not have updates of the form $x :\sim y + c$ (with $\sim \in \{<, \leq, \geq, >\}$).

6.5 Summary of the Expressiveness Results

In this section, we proved the expressiveness results which are summarized in Table 3 (TA represents the class ***Uta***$(\mathcal{C}(X),\mathcal{U}_0(X))$ whereas TA$_\varepsilon$ represents the class ***Uta***$_\varepsilon(\mathcal{C}(X),\mathcal{U}_0(X))$. The sign $>_\ell$ means "strictly more expressive" (from a language point of view).

<table>
<tr><th></th><th>$\mathcal{U}_0(X) \cup ...$</th><th>Diagonal-free constraints</th><th>General constraints</th></tr>
<tr><td>1</td><td>$x := c, x := y$</td><td rowspan="3">$\equiv_s$ TA</td><td>$\equiv_s$ TA</td></tr>
<tr><td>2</td><td>$x := x + 1$</td><td rowspan="3">Turing</td></tr>
<tr><td>3</td><td>$x := y + c$</td></tr>
<tr><td>4</td><td>$x := x - 1$</td><td>Turing</td></tr>
<tr><td>5</td><td>$x :< c$</td><td rowspan="4">TA$_\varepsilon$</td><td>$>_\ell$ TA, TA$_\varepsilon$</td></tr>
<tr><td>6</td><td>$x :> c$</td><td rowspan="4">Turing</td></tr>
<tr><td>7</td><td>$x :\sim y + c$</td></tr>
<tr><td>8</td><td>$y + c <: x :< y + d$</td></tr>
<tr><td>9</td><td>$y + c <: x :< z + d$</td><td>Turing</td></tr>
</table>

with $\sim \in \{\leq, <, >, \geq\}$ and $c,\ d \in \mathbb{Q}^+$

Table 3. Expressiveness results

The updatable timed automata model is thus not much more expressive than classical timed automata. The transformation of a (decidable) updatable timed automaton into a classical timed automaton with silent actions suffers from a big combinatorics blow-up, thus updates appear to provide a **synthetic** way to represent timed behaviours. We do not know whether some simpler transformation exists, but the preliminary examples 13, 14 and 15 let us think that it is rather improbable that it exists.

7 Conclusion

In this paper, we studied a natural extension of Alur and Dill's timed automata, based on the possibility to update clocks in a more elaborate way than simply reset them to zero. Our results concern both decidability results (summarized in table 2, page 25) and expressiveness properties (summarized in table 3, page 43).

Our work lets open some mostly theoretical questions about updatable timed automata. For example, one could be interested in the following questions:

- Is it possible to transform an updatable timed automaton into an equivalent traditional timed automaton in a simpler way than the one presented in section 6?
- Is it sometimes unavoidable to use ε-transitions when transforming updatable timed automata into equivalent timed automata? If so, when can we do so?

However, from our point of view, the main interest of this work is to provide a sound theoretical framework for the use of updatable timed automata as a model in real case studies (if that was necessary, a recent paper [Bou03] has recalled how much these theoretical frameworks were necessary to tools). Indeed, updatable timed automata allow to represent in a concise way systems which can not be modelled in a natural way by timed automata. We also proved that analyzing these models can be done in a complexity not higher than the one of classical timed automata. Subclasses of updatable timed automata have been implemented in the tool UPPAAL. Their implementation uses a technique inspired by our Diophantine inequations systems [BBFL03].

Acknowledgements: We would like to thank Béatrice Bérard for her careful reading of the paper and her comments.

References

[ACD+92] Rajeev Alur, Costas Courcoubetis, David Dill, Nicolas Halbwachs, and Howard Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proc. 13th IEEE Real-Time Systems Symposium (RTSS'92)*, pages 157–166. IEEE Computer Society Press, 1992.

[ACH94] Rajeev Alur, Costas Courcoubetis, and Thomas A. Henzinger. The observational power of clocks. In *Proc. 5th International Conference on Concurrency Theory (CONCUR'94)*, volume 836 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 1994.

[AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[AD94] Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science (TCS)*, 126(2):183–235, 1994.

[AFH94] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. A determinizable class of timed automata. In *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Vardi. Parametric real-time reasoning. In *Proc. 25th Annual ACM Symposium on the Theory of Computing (STOC'93)*, pages 592–601. ACM, 1993.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim G. Larsen. Static guard analysis in timed automata verification. In *Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003)*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer, 2003.

[BBP02] Béatrice Bérard, Patricia Bouyer, and Antoine Petit. Analysing the PGM protocol with UPPAAL. In *Proc. 2nd Workshop on Real-Time Tools (RT-TOOLS'02)*, 2002. Proc. published as Technical Report 2002-025, Uppsala University, Sweden.

[BD00] Béatrice Bérard and Catherine Dufourd. Timed automata and additive clock constraints. *Information Processing Letters (IPL)*, 75(1–2):1–7, 2000.

[BDFP00a] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are timed automata updatable? In *Proc. 12th International Conference on Computer Aided Verification (CAV'2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2000.

[BDFP00b] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Expressiveness of updatable timed automata. In *Proc. 25th International Symposium on Mathematical Foundations of Computer Science (MFCS'2000)*, volume 1893 of *Lecture Notes in Computer Science*, pages 232–242. Springer, 2000.

[BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2–3):145–182, 1998.

[BF99] Béatrice Bérard and Laurent Fribourg. Automated verification of a parametric real-time program: the ABR conformance protocol. In *Proc. 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 1999.

[BFKM03] Béatrice Bérard, Laurent Fribourg, Francis Klay, and Jean-François Monin. A compared study of two correctness proofs for the standardized algorithm of abr conformance. *Formal Methods in System Design*, 22(1):59–86, 2003.

[BGP96] Béatrice Bérard, Paul Gastin, and Antoine Petit. On the power of non-observable actions in timed automata. In *Proc. 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS'96)*, volume 1046 of *Lecture Notes in Computer Science*, pages 257–268. Springer, 1996.

[BLL+98] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi, and Carsten Weise. New generation of UPPAAL. In *Proc. International Workshop on Software Tools for Technology Transfer (STTT'98)*, BRICS Notes Series, pages 43–52, 1998.

[Bou03] Patricia Bouyer. Untameable timed automata! In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631. Springer, 2003.

[CG00] Christian Choffrut and Massimiliano Goldwurm. Timed automata with periodic clock constraints. *Journal of Automata, Languages and Combinatorics (JALC)*, 5(4):371–404, 2000.

[DGP97] Volker Diekert, Paul Gastin, and Antoine Petit. Removing ε-transitions in timed automata. In *Proc. 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS'97)*, volume 1200 of *Lecture Notes in Computer Science*, pages 583–594. Springer, 1997.

[Dom91] Eric Domenjoud. Solving systems of linear diophantine equations: An algebraic approach. In *Proc. 16th International Symposium on Mathematical Foundations of Computer Science (MFCS'91)*, volume 520 of *Lecture Notes in Computer Science*, pages 141–150. Springer, 1991.

[DOTY96] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool KRONOS. In *Proc. Hybrid Systems III: Verification and Control (1995)*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1996.

[DOY94] Conrado Daws, Alfredo Olivero, and Sergio Yovine. Verifying et-lotos programs with KRONOS. In *Proc. 7th International Conference on Formal Description Techniques (FORTE'94)*, pages 227–242. Chapman & Hall, 1994.

[Duf97] Catherine Dufourd. Une extension d'un résultat d'indécidabilité pour les automates temporisés. In *Proc. 9th Rencontres Francophones du Parallélisme (RenPar'97)*, 1997.

[DZ98] François Demichelis and Wieslaw Zielonka. Controlled timed automata. In *Proc. 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 1998.

[HHWT95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HYTECH. In *Proc. 1st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer, 1995.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model-checker for hybrid systems. *Journal on Software Tools for Technology Transfer (STTT)*, 1(1–2):110–122, 1997.

[HKWT95] Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. The expressive power of clocks. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP'95)*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer, 1995.

[HSLL97] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 2–13. IEEE Computer Society Press, 1997.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[JLS96] Henrik E. Jensen, Kim G. Larsen, and Arne Skou. Modelling and analysis of a collision avoidance protocol using SPIN and UPPAAL. In *Proc. 2nd SPIN Verification Workshop on Algorithms, Applications, Tool Use, Theory*. American Mathematical Society, 1996.

[LL98] François Laroussinie and Kim G. Larsen. Cmc: A tool for compositional model-checking of real-time systems. In *Proc. IFIP Joint International Conference on Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pages 439–456. Kluwer Academic, 1998.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer (STTT)*, 1(1–2):134–152, 1997.

[Mil89] Robert Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[Min67] Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall International, 1967.

[Par81] David Park. Concurrency and automata on infinite sequences. In *Proc. 5th Conference on Theoretical Computer Science (TCS'81)*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.

[Wil94] Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proc. 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, volume 863 of *Lecture Notes in Computer Science*, pages 694–715. Springer, 1994.

[Yi90] Wang Yi. Real-time behaviour of asynchronous agents. In *Proc. 1st International Conference on Theory of Concurrency (CONCUR'90)*, volume 458 of *Lecture Notes in Computer Science*, pages 502–520. Springer, 1990.

[Yi91] Wang Yi. *A Calculus of Real-Time Systems*. PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 1991.

[Yov97] Sergio Yovine. KRONOS: A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer (STTT)*, 1(1–2):123–133, 1997.