Journal of Automata, Languages and Combinatorics **u** (v) w, x–y

On Conciseness of Extensions of Timed Automata

Patricia Bouyer and Fabrice Chevalier
LSV – CNRS & ENS de Cachan
61, avenue du Président Wilson, 94230 Cachan, France
e-mail: {bouyer,chevalie}@lsv.ens-cachan.fr

ABSTRACT
In this paper we study conciseness of various extensions of timed automata, and prove that several features like diagonal constraints or updates lead to exponentially more concise timed models.

Keywords: timed automata, conciseness, diagonal constraints, updates

1. Introduction

Model-checking of real-time systems. Since their introduction by Alur and Dill in the beginning of the 90's [2, 3], timed automata are a widely used and studied model for real-time systems. Since that time, numerous works have been devoted to the theoretical comprehension of this model. The success of this model is mostly due to the decidability of reachability properties [3] and to its use in verification, and more specifically in model-checking [1]. It is worth to notice that based on these decidability properties, several tools analyzing timed automata have been developed [9, 14] and successfully used.

Modelling real-time systems. With the motivation to easily model numerous real systems, many extensions of timed automata have been proposed and studied, among others extensions of clock constraints [5, 8], extensions of operations on clocks [6], use of silent actions [4], variation of the slopes of clocks [12, 10, 7]. For such extensions, several points are of interest: a) the decidability of reachability properties (to use this model for verification purposes), b) the expressive power (to model a large class of systems), and c) the conciseness of induced models. The first two points are classical when studying models for verification purposes: it is fundamental to have models which can be analyzed and which can represent many real systems (indeed models used for verification have to be the result of a compromise between expressiveness and decidability). Point c) is discussed below.

Conciseness of models. It is most of the time simpler to use high-level features when modelling real systems (as it is easier to use advanced programming languages than assembly languages when writing programs). Induced models are often smaller, which makes modelling less error-prone. Conciseness of features and models is then an important property. For example it has been proved that LTL with past is as expressive as classical LTL, but is *exponentially more concise* [15]. This means that there are formulas of LTL with past which can only be expressed as formulas in LTL at an exponential cost in the size of the formulas. On the other hand model-checking LTL with past is not more difficult than model-checking LTL [16]. Using past modalities should then always be considered when writing formulas. In the timed framework, not many works can be found which report conciseness results, the only related work is the one on concurrent timed automata [13] where relative conciseness of classes of concurrent timed automata are established.

Contributions of this paper. Diagonal constraints (*i.e.* constraints of the form $x - y \bowtie c$) are commonly used features, and are known to not add expressive power to classical timed automata [3]. The classical construction for removing diagonal constraints in a timed automaton [4] nevertheless induces an exponential blowup in the size of the model. Even though it seems that this construction can not be improved, it has not been proved yet that this exponential blowup is unavoidable, or if a more clever construction for removing diagonal constraints can be found. In this paper, we prove that this is indeed the case, and that models using diagonal constraints are exponentially more concise than models using only diagonal-free constraints.

Updates of clocks have been studied in [6], and though the general model of updatable timed automata is undecidable, several decidable subclasses have been described. For each such subclass, a construction removing updates (thus transforming updatable timed automata into classical timed automata possibly with silent actions) is proposed, and as for the diagonal constraints, the construction suffers from an exponential blowup in the size of the model, and it was conjectured as unavoidable. In this paper we prove this conjecture and state that all decidable subclasses of updatable timed automata described in [6] are indeed exponentially more concise than classical timed automata (possibly with silent actions).

All our conciseness proofs rely on the fact that with a linear number (in some integer n) of clocks, when using features like diagonal constraints or updates, it is possible to count up to 2^n with no much structure in the automaton, which can not be the case with only constraints of the form $x \bowtie c$ and resets to 0.

Plan of the paper. The paper is organized as follows. In Section 2 we present all basic material that we will need in the following. In Section 3 we present in detail the conciseness proof for timed automata using diagonal clock constraints. In Section 4 we present the conciseness proofs for updatable timed automata, by distinguishing the nature of updates which are used: we first study in subsection 4.1 timed automata using deterministic updates, and then in subsection 4.2 timed automata using non-deterministic updates.

2. Preliminaries

Timed words and clocks. If Z is any set, let Z^* be the set of finite sequences of elements in Z. 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 non decreasing sequence $\tau = (t_i)_{1\leq i\leq n} \in \mathbb{T}^*$. A *timed word* $\omega = (a_i, t_i)_{1\leq i\leq n}$ is an element of $(\Sigma \times \mathbb{T})^*$, also written as a pair $\omega = (\sigma, \tau)$, where $\sigma = (a_i)_{1\leq i\leq n}$ is a word in Σ^* and $\tau = (t_i)_{1\leq i\leq n}$ a time sequence in $\mathbb{T}^*$ 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 $(v + t)(x) = v(x) + t$, $\forall x \in X$.

Clock constraints. Given a set of clocks X, we introduce two sets of clock constraints over X. The most general one, denoted by $\mathcal{C}(X)$ or simply $\mathcal{C}$, is defined by the grammar $g ::= x \bowtie c \mid x-y \bowtie c \mid g\wedge g$ where $x, y \in X$, $c \in \mathbb{Q}$, and $\bowtie \in\{\leq, <, =, >, \geq\}$.

We also consider the proper subset of *diagonal-free* clock constraints, denoted $\mathcal{C}_{df}(X)$ or simply $\mathcal{C}_{df}$, where constraints of the form $x - y \bowtie c$ are not allowed. Note that this restricted set of constraints is called diagonal-free because constraints of the form $x - y \bowtie c$ are called *diagonal* clock constraints.

Clock constraints are interpreted over clock valuations. The satisfaction relation, denoted as "$v \models g$" (g being a clock constraint and v a valuation), is defined by: $v \models x \bowtie c$ if $v(x) \bowtie c$, $v \models x - y \bowtie c$ if $v(x) - v(y) \bowtie c$, and $v \models g_1 \wedge g_2$ if $v \models g_1$ and $v \models g_2$. If g is a clock constraint, we note $[\![g]\!]$ the set of valuations satisfying g.

Updates. An *update* is a function from $\mathbb{T}^X$ to $\mathcal{P}(\mathbb{T}^X)$ which assigns to each valuation a set of valuations. In this work, updates are restricted according to the following definitions.

A *simple update* over a clock z has one of the two following forms: $up ::= z :\bowtie c \mid z :\bowtie x + c$ where $c \in \mathbb{Q}$, $x, z \in X$, and $\bowtie \in\{\leq, <, =, >, \geq\}$. Let v be a valuation and up 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 $v'(z) \bowtie c$ (resp. $v'(z) \bowtie v(x)+c$) whenever up is $z :\bowtie c$ (resp. $z :\bowtie x+c$)

In what follows, an *update* over a set of clocks X is a collection $up = (up_i)_{1\leq i\leq k}$, where each up_i is a simple update over some clock $x_i \in X$. Let v, $v' \in \mathbb{T}^n$ be two clock valuations. Then we say that $v' \in up(v)$ if for all i, the clock valuation v_i defined by $v_i(x_i) = v'(x_i)$ and $v_i(y) = v(y)$ for any $y \neq x_i$ is in $up_i(v)$. The set of updates over the set of clocks X is denoted by $\mathcal{U}(X)$.

In classical timed automata [3], updates which are used are only *resets*, that is updates of the form $z := 0$. We denote the set of resets by $\mathcal{U}_0(X)$ (or simply $\mathcal{U}_0$ when X is clear from the context). General updates have been studied in [6].

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 & Dill's timed automata [2, 3].

An *updatable timed automaton* over $\mathbb{T}$ is a tuple $\mathcal{A} = (\Sigma, X, Q, T, I, F)$, 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, and $F \subseteq Q$ is the subset of final 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*. Such actions will be interpreted in a classical way, see [4] for formal definitions.

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 $\boldsymbol{Uta}_\varepsilon(\mathcal{C},\mathcal{U})$ 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 $\boldsymbol{Uta}(\mathcal{C},\mathcal{U})$. Classical Alur & Dill's timed automata correspond to the class $\boldsymbol{Uta}(\mathcal{C}_{df}(X),\mathcal{U}_0(X))$.

As for timed automata, a behaviour in an updatable timed automaton is obtained through the notion of paths and runs. Let us fix for the rest of this section an updatable timed automaton $\mathcal{A}$. A *path* in $\mathcal{A}$ is a finite sequence of consecutive transitions $P = q_0 \xrightarrow{g_1,a_1,up_1} q_1 \ldots \xrightarrow{g_p,a_p,up_p} q_p$ where $(q_{i-1}, g_i, a_i, up_i, q_i) \in T$ for all $i > 0$. The path is said to be *accepting* if it starts in an initial state ($q_0 \in I$) and ends in a final state ($q_p \in F$).

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 $\langle q_0, v_0\rangle \xrightarrow[\tau_1]{a_1} \langle q_1, v_1\rangle \ldots \xrightarrow[\tau_p]{a_p} \langle q_p, v_p\rangle$ where $\tau = (\tau_i)_{i\geq 0}$ is a time sequence and $(v_i)_{i\geq 0}$ are clock valuations such that $\begin{cases} v_{i-1} + (\tau_i - \tau_{i-1}) \models \varphi_i \\ v_i \in up_i\,(v_{i-1} + (\tau_i - \tau_{i-1})) \end{cases}$ with $\tau_0 = 0$. Note that any set $up_i(v_{i-1} + (\tau_i - \tau_{i-1}))$ of a run has to be non empty.

The label of such a run is the timed word $w = (a_1, \tau_1)(a_2, \tau_2) \ldots$ 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 $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

Conciseness of classes of automata. Let $\mathcal{A}$ be a timed automaton. The size of $\mathcal{A}$, denoted $\mathsf{Size}(\mathcal{A})$, is the length of its encoding (states and transitions) on the tape of a Turing Machine (in particular we suppose a binary encoding for constants).

Let $\mathcal{S}$ and $\mathcal{S}'$ be two classes of timed automata. The class $\mathcal{S}$ is said *exponentially more concise* than the class $\mathcal{S}'$ whenever there exists a sequence of timed automata $(\mathcal{A}_n)_{n\geq 0}$ in $\mathcal{S}$ of polynomial size in n such that for any sequence of timed automata $(\mathcal{B}_n)_{n\geq 0}$ such that $L(\mathcal{A}_n) = L(\mathcal{B}_n)$, $\mathsf{Size}(\mathcal{B}_n)$ is at least exponential in n.

3. Conciseness of Timed Automata with Diagonal Constraints

Diagonal constraints have been introduced in the seminal paper [3]. It's a folklore knowledge that timed automata with diagonal constraints are not more expressive than diagonal-free timed automata. This is formally stated by the following proposition, whose proof can be found in [4].

Proposition 1 *For every timed automaton $\mathcal{A}$ in the class $\boldsymbol{Uta}(\mathcal{C},\mathcal{U}_0)$, there exists a timed automaton $\mathcal{B}$ in $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_0)$ such that $L(\mathcal{A}) = L(\mathcal{B})$.*

The construction presented in [4] suffers from an exponential blowup of the number of states. The question of whether this exponential blowup is unavoidable or not was an open problem, even if intuition was that this blowup could not be avoided. Conciseness results are known in the context of concurrent timed automata [13], but they do not apply to classical timed automata as concurrent timed automata are already exponentially more succinct than timed automata. We answer positively to the aforementioned intuition and prove the following result:

Theorem 2 *The class $\boldsymbol{Uta}(\mathcal{C},\mathcal{U}_0)$ is exponentially more concise than the class $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_0)$.*

To prove this theorem we define a family of languages $(L_n)_{n\geq 0}$ as follows:

$$L_n = \{(a^{2^n},\tau) \mid 0 < \tau_1 < \tau_2 < \cdots < \tau_{2^n} < 1\}$$

We then prove two properties: (i) we can construct timed automata $(\mathcal{A}_n)_{n\geq 0}$ in $\boldsymbol{Uta}(\mathcal{C},\mathcal{U}_0)$ such that $L(\mathcal{A}_n) = L_n$ and $\mathsf{Size}(\mathcal{A}_n) \in \mathcal{O}(n^2.\log n)$, and (ii) for every timed automaton $\mathcal{B}_n$ in $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_0)$ such that $L(\mathcal{B}_n) = L_n$, $\mathsf{Size}(\mathcal{B}_n) \geq 2^n$.

Lemma 3 *For every $n \geq 0$, the language L_n is recognized by a timed automaton $\mathcal{A}_n$ of $\boldsymbol{Uta}(\mathcal{C},\mathcal{U}_0)$ such that $\mathsf{Size}(\mathcal{A}_n) \in \mathcal{O}(n^2.\log n)$.*

Proof. Fix an integer $n \geq 0$. We construct a timed automaton $\mathcal{A}_n$ with diagonal constraints which has 2 states, $2n+2$ clocks (denoted $\{x_i, x'_i\}_{1\leq i\leq n}$, y and z), $n+1$ transitions, and which uses constants 0 and 1. The idea of the construction is to encode with clocks a binary counter c. This counter will take values between 0 and $2^n - 1$ and will count the number of a's which are done along a computation. The binary encoding $b_1 b_2 \cdots b_n$ of c (least significant bit first) will be related to the value of $\mathcal{A}$'s clocks in the following way: $b_i = 1$ if $x_i - x'_i > 0$ while $b_i = 0$ if $x_i - x'_i = 0$ (these two conditions are invariant when time elapses). Automaton $\mathcal{A}_n$ will then increment counter c each time an action a is done, until value of c reaches $2^n - 1$, and do a last a. The general shape of $\mathcal{A}_n$ is depicted on Figure 1. Now we just need to explain how we can increment and test counter c. We will ensure that the time sequence is increasing using clock y which will be checked ($y > 0$) and reset on every transition. We will ensure that the global time is bounded by 1 by adding a constraint $z < 1$ on every transition (this clock z will never be reset). We now abstract away clocks y and z and concentrate on the encoding of counter c.

Incrementing counter c encoded by $b_1 b_2 \cdots b_n$ can be done as follows: if the $j-1$ first bits are equal to 1 and if the j^{th} bit is 0, then we only need to set bits $(b_i)_{1\leq i\leq j-1}$ to 0 and bit b_j to 1. Transitions we use to increment c are then the following: for every $1 \leq j \leq n$ there is a loop on state ℓ of $\mathcal{A}_n$ labelled by (g_j, a, Y_j) where

$$\begin{cases} g_j \text{ is } \bigwedge_{i=1}^{j-1}(x_i - x'_i > 0) \wedge (x_j - x'_j = 0) \\ Y_j \text{ is } \bigcup_{i=1}^{j-1}\{x_i, x'_i\} \cup \{x'_j\} \end{cases}$$

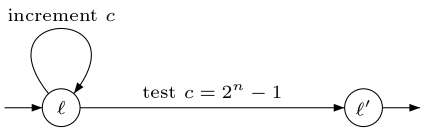


Figure 1: Shape of automaton recognizing L_n

Constraint g_j precisely checks that the $j-1$ first bits are 1 whereas the j^{th} bit is 0. Resetting both clocks x_i and x'_i ensures that the constraint $x_i - x'_i = 0$ will be satisfied (this encodes $b_i = 0$). On the other hand, resetting clock x'_j but not x_j will ensure the constraint $x_j - x'_j > 0$ as time is strictly increasing between two a's. The value of all bits b_i (for $i > j$) remains unchanged.

When counter c equals $2^n - 1$ and an a is read, automaton $\mathcal{A}_n$ leaves state ℓ and goes to ℓ': there is a transition from ℓ to ℓ' labelled $(g', a, \varnothing)$ where g' is the constraint $\bigwedge_{i=1}^{n}(x_i - x'_i > 0)$.

Note that automaton $\mathcal{A}_n$ is deterministic. It is then easy to check that $\mathcal{A}_n$ recognizes the language L_n: if the timed word (a^p, τ) can be read in $\mathcal{A}_n$, then τ is a strictly increasing time sequence bounded by 1, and two cases may arise:

- either $p < 2^n$, and the current configuration of $\mathcal{A}_n$ is (ℓ, v_p) where v_p is a valuation of the clocks corresponding to the binary encoding $b_1 b_2 \cdots b_n$ of p: $v_p(x_i) - v_p(x'_i) = 0$ if $b_i = 0$ and $v_p(x_i) - v_p(x'_i) > 0$ if $b_i = 1$.
- or $p = 2^n$, and final state has been reached.

We now evaluate the size of $\mathcal{A}_n$: there are n loops on state ℓ and one transition from state ℓ to ℓ'. On one loop (or transition), there are at most n diagonal constraints, each constraint can be encoded using $\mathcal{O}(\log n)$ space. The size of $\mathcal{A}_n$ is thus in $\mathcal{O}(n^2 . \log n)$. □

Lemma 4 *For every $n \geq 0$, if $\mathcal{B}_n$ is a timed automaton of* $\mathbf{Uta}(\mathcal{C},\mathcal{U}_0)$ *which recognizes L_n, then* $\mathsf{Size}(\mathcal{B}_n) \geq 2^n$.

Proof. We show that $\mathcal{B}_n$ has at least 2^n states by contradiction. Suppose that $\mathcal{B}_n$ has strictly less than 2^n states, and note m the smallest positive constant among 1 and constants appearing in some constraint of $\mathcal{B}_n$. Consider a timed word $w = (a^{2^n}, \tau)$ such that $0 < \tau_1 < \cdots < \tau_{2^n} < m$. This word is accepted by $\mathcal{B}_n$ along a path $P = q_0 \xrightarrow{g_1,a,Y_1} q_1 \ldots \xrightarrow{g_{2^n},a,Y_{2^n}} q_{2^n}$. We assume that the run accepting w through P is $\langle q_0, v_0\rangle \xrightarrow[\tau_1]{a} \langle q_1, v_1\rangle \ldots \xrightarrow[\tau_{2^n}]{a} \langle q_{2^n}, v_{2^n}\rangle$. For every $0 \leq i < 2^n$, for every clock x, $v_i(x) + \tau_{i+1} - \tau_i \in]0, m[$, which implies that $]0, m[^X \subseteq [\![g_i]\!]$. As $\mathcal{B}_n$ has strictly less than 2^n states, there exist $0 \leq i < j \leq 2^n$ such that $q_i = q_j$. Consider a timed word $w' = (a^{2^n-(j-i)}, \tau')$ such that $0 < \tau'_1 < \cdots < \tau'_{2^n-(j-i)} < m$. Using the fact that $]0, m[^X \subseteq [\![g]\!]$ for every guard g appearing along path P, we get that there is an accepting run for w' through the path $P' = q_0 \xrightarrow{g_1,a,Y_1} q_1 \ldots \xrightarrow{g_i,a,Y_i} q_i = q_j \xrightarrow{g_{j+1},a,Y_{j+1}} \ldots q_{2^n}$. We thus get that $\mathcal{B}_n$ accepts a word of length strictly less than 2^n, which can not be the case. This concludes the proof of Lemma 4. □

Remark. If we now consider the language

$$\{(a^i, \tau) \mid 2^n \leq i < 2^{n+1} \text{ and } 0 < \tau_1 < \cdots < \tau_i < 1\}$$

which is a slight modification of L_n, we can prove that the construction of [4] is "optimal": the size of any timed automaton from $\textit{\textbf{Uta}}(\mathcal{C}_{df}, \mathcal{U}_0)$ recognizing the language above has at least 2^{n+1} states, while it can be recognized by a timed automaton from $\textit{\textbf{Uta}}(\mathcal{C}, \mathcal{U}_0)$ with only two states and n different diagonal constraints. Removing diagonal constraints in the previous automaton using the construction in [4] leads to a timed automaton from $\textit{\textbf{Uta}}(\mathcal{C}_{df}, \mathcal{U}_0)$ with 2^{n+1} states.

4. Conciseness of Updatable Timed Automata

Decidability and expressiveness of updatable timed automata have been studied in [6]. Though the whole class of updatable timed automata is undecidable, several subclasses have been proved decidable (a class is said *decidable* if the emptiness problem (or equivalently the reachability problem) for this class is decidable). For every timed automaton in those subclasses, it has been proved that there exists a classical timed automaton (possibly with ε-transitions) which recognizes the same timed language. Constructions presented in [6] suffer however from an exponential blowup of the number of states, and it was conjectured that this exponential blowup was unavoidable. In this section we prove this conjecture. We now distinguish two cases, depending on the nature of updates which are used.

4.1. Case of Deterministic Updates

We define several sets of updates: (i) $\mathcal{U}_{x:=y}$ is the set of updates of the form $x := y$ and of resets of clocks, (ii) $\mathcal{U}_{x:=1}$ is the set of updates of the form $x := 1$ and of resets of clocks, and (iii) $\mathcal{U}_{x:=y+1}$ is the set of updates of the form $x := y + 1$ and of resets of clocks.

Proposition 5 ([6]) *The three classes* $\textbf{Uta}(\mathcal{C}_{df}, \mathcal{U}_{x:=y})$, $\textbf{Uta}(\mathcal{C}_{df}, \mathcal{U}_{x:=1})$, $\textbf{Uta}(\mathcal{C}_{df}, \mathcal{U}_{x:=y+1})$ *are decidable. Moreover, for every timed automaton* $\mathcal{A}$ *in one of the classes* $\textbf{Uta}(\mathcal{C}_{df}, \mathcal{U}_{x:=y})$, $\textbf{Uta}(\mathcal{C}_{df}, \mathcal{U}_{x:=1})$, *or* $\textbf{Uta}(\mathcal{C}_{df}, \mathcal{U}_{x:=y+1})$, *there exists a timed automaton* $\mathcal{B}$ *in* $\textbf{Uta}(\mathcal{C}_{df}, \mathcal{U}_0)$ *such that* $L(\mathcal{A}) = L(\mathcal{B})$.

As in the previous section, the above-mentioned timed automaton $\mathcal{B}$ suffers from an exponential blowup of the number of states. The next theorems state that this exponential blowup is indeed unavoidable.

Theorem 6 *The two classes* $\textbf{Uta}(\mathcal{C}_{df}, \mathcal{U}_{x:=1})$ *and* $\textbf{Uta}(\mathcal{C}_{df}, \mathcal{U}_{x:=y+1})$ *are exponentially more concise than* $\textbf{Uta}(\mathcal{C}_{df}, \mathcal{U}_0)$.

Proof. In both cases we use the sequence $(L_n)_{n \geq 0}$ of languages already considered in Section 3. In Lemma 4 we have shown that any timed automaton in $\textit{\textbf{Uta}}(\mathcal{C}_{df}, \mathcal{U}_0)$ which recognizes L_n has at least 2^n states. We thus only need to show that L_n can be recognized by timed automata in $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:=1})$ (resp. in $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:=y+1})$) of polynomial size. We use the same technics as the ones used in the proof of Lemma 3: we encode with clocks a binary counter which counts the number of a's which are read.

The shape of automaton $\mathcal{A}_n$ we will construct is the same as the one used in the proof of Lemma 3 (depicted on Figure 1). Automaton $\mathcal{A}_n$ has 2 states, $n+2$ clocks (denoted $\{x_i\}_{i=1..n}$, y and z), $n+1$ transitions and uses constants 0 and 1. Clocks y and z are used in a similar way as in the proof of Lemma 3. The binary encoding $b_1 b_2 \cdots b_n$ of the counter is done using clocks as follows: the value of bit b_i is encoded using a single clock x_i, and $b_i = 0$ if $x_i < 1$ whereas $b_i = 1$ if $x_i \geq 1$. To set bit b_i to 1, we use update $x_i := 1$ (resp. $x_i := x_i + 1$), whereas to set bit b_i to 0, we use in both cases the reset $x_i := 0$. It is then easy to implement the increment of the counter and to test the value of the counter. We do not enter into more details, and only mention that the size of the automaton we construct is in $\mathcal{O}(n^2.\log n)$. □

To show that the class $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:=y})$ is exponentially more concise that classical timed automata, we have to use another method, as languages $(L_n)_{n\geq 0}$ cannot be recognized by timed automata of polynomial size even using updates of the form $x := y$. The intuition is that, even if we can use updates $x := y$, if time elapsed is small, all values of clocks will remain small (which is for example not the case with updates of the form $x := 1$). The following theorem is however true, but this will require a more involved proof.

Theorem 7 *The class* $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:=y})$ *is exponentially more concise than the class* $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_0)$.

We now consider the timed languages $(L'_n)_{n\geq 0}$ where

$$L'_n = \{(a^{2^n}, \tau) \mid 1 \leq \tau_1 \leq ... \leq \tau_{2^n} < 2\}$$

The proof of Theorem 7 then relies on the two following lemmas.

Lemma 8 *For every* $n \geq 0$*, there exists a timed automaton* $\mathcal{A}_n$ *in* $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:=y})$ *which recognizes* L'_n *such that* $\mathsf{Size}(\mathcal{A}_n)$ *is in* $\mathcal{O}(n^2.\log n)$.

Proof. As before we use a binary counter c encoded with n clocks $(x_i)_{i=1..n}$. The binary encoding of counter c is written as $b_1 b_2 \cdots b_n$, and we will have that $b_i = 1$ whenever the value of x_i is greater than 1, and we will have that $b_i = 0$ whenever the value of x_i is strictly less than 1. We also use another clock y which is never reset and which will be used to check that the time sequence is between 1 and 2. The shape for an automaton recognizing L'_n is the same as the one for L_n and is thus depicted on Figure 1; we only need to add a constraint $1 \leq y < 2$ on every transition of the automaton. It remains to explain how we increment a bit of the encoding of counter c. To set b_i to 1 we use the update $x_i := y$ (as the value of y is always in the interval $[1, 2[$, the new value of x_i will also be in $[1, 2[$) and to set b_i to 0 we reset clock x_i. Incrementing counter c or testing the value of counter c is done in a similar way as in Lemma 3 (propagating the carry). The size of the resulting automaton is also in $\mathcal{O}(n^2.\log n)$. $\square$

We now show that if $\mathcal{B}_n$ is a timed automaton of $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_0)$ which recognizes L'_n, then $\mathsf{Size}(\mathcal{B}_n) \geq 2^n$. Contrary to Lemma 4 it may be the case that $\mathcal{B}_n$ has less than 2^n states. Indeed there exists a timed automaton with two states but $2^n - 1$ clocks which recognize L'_n: each time an a is done, one clock is reset, and the execution terminates and goes to final state when all clocks are below 1 (which means that all clocks have been reset once, and thus that a^{2^n-1} have already be done. We will then prove the following lemma which takes into account both the number of states and the number of clocks used in the automaton.

Lemma 9 *Let $n \geq 0$. If $\mathcal{B}_n$ is a timed automaton of $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_0)$ with k clocks and l states which recognizes L'_n, then $(k+1)l > 2^n$.*

Proof. We show this lemma by contradiction. Suppose that $(k+1)l \leq 2^n$ and note m the common denominator of constants appearing in $\mathcal{B}_n$. Consider a word $w = (a^{2^n}, \tau)$ such that $1 < \tau_1 < \tau_2 < \cdots < \tau_{2^n} < 1 + \frac{1}{m}$. Let $r = \langle q_0, v_0\rangle \xrightarrow[t_1]{a} \langle q_1, v_1\rangle \ldots \xrightarrow[t_{2^n}]{a} \langle q_p, v_{2^n}\rangle$ be an accepting run for w. First notice that for all index i and clock x, $v_i(x)$ is either smaller than $\frac{1}{m}$ or between 1 and $1+\frac{1}{m}$. From this remark we associate to each valuation v_i a vector in $\{0,1\}^k$, denoted $\pi(v_i)$, such that $\pi(v_i)_h = 0$ if $v_i(x_h) \in [0, \frac{1}{m}[$ and $\pi(v)_h = 1$ if $v_i(x_h) \in]1, 1+\frac{1}{m}[$. It is easy to see that for each $1 \leq i < 2^n$, $\pi(v_{i+1}) \leq \pi(v_i)$ (in the sense that for every $1 \leq h \leq k$, $\pi(v_{i+1})_h \leq \pi(v_i)_h$). As $\pi(v_i)$ is decreasing, there are at most $(k+1)l$ possible pairs $(q_i, \pi(v_i))$ along run r. Since we have assumed that $(k+1)l \leq 2^n$, there exist two indices $i < j$ such that $(q_i, \pi(v_i)) = (q_j, \pi(v_j))$. As in the proof of Lemma 4, we can remove the transitions in-between states q_i and q_j and get an accepting run with length less than 2^n, which is not possible. This concludes the proof of Lemma 9. $\square$

It then follows that every automaton $\mathcal{B}_n$ which recognizes L'_n is at least of size $2^{\frac{n}{2}}$, which concludes the proof of Theorem 7.

4.2. Case of Non-Deterministic Updates

We distinguish several sets of non-deterministic updates: (i) $\mathcal{U}_{x:<1}$ is the set of updates which allow updates of the form $x :< 1$ and resets of clocks, (ii) $\mathcal{U}_{x:>1}$ is the set of updates which allow updates of the form $x :> 1$ and resets of clocks, (iii) $\mathcal{U}_{x:<y}$ is the set of updates which allow updates of the form $x :< y$ and resets of clocks, and (iv) $\mathcal{U}_{x:>y}$ is the set of updates which allow updates of the form $x :> y$ and resets of clocks.

Proposition 10 ([6]) *For every timed automaton $\mathcal{A}$ in one of the classes $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:<1})$, $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:>1})$, $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:<y})$ or $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:>y})$ there exists a timed automaton $\mathcal{B}$ in $\boldsymbol{Uta}_\varepsilon(\mathcal{C}_{df},\mathcal{U}_0)$ such that $L(\mathcal{A}) = L(\mathcal{B})$.*

As previously, the above-mentioned timed automaton $\mathcal{B}$ suffers from an exponential blowup of the number of states, which we will prove is unavoidable.

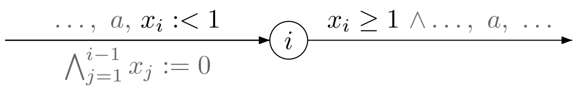
Theorem 11 *The four classes* $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:<1})$, $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:>1})$, $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:<y})$ *and* $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:>y})$ *are exponentially more concise than* $\boldsymbol{Uta}_\varepsilon(\mathcal{C}_{df},\mathcal{U}_0)$.

We will prove that $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:<1})$ is exponentially more concise than $\boldsymbol{Uta}_\varepsilon(\mathcal{C}_{df},\mathcal{U}_0)$, similar proofs can be written for the other classes. We will use the languages $L_n = \{(a^{2^n},\tau) \mid 0 < \tau_1 < \tau_2 < \cdots < \tau_{2^n} < 1\}$. The proof of Theorem 11 then relies on the two following lemmas.

Lemma 12 *For every* $n \geq 0$, *there exist timed automata* $(\mathcal{A}_n)_{n\geq 0}$ *in* $\boldsymbol{Uta}(\mathcal{C}_{df},\mathcal{U}_{x:<1})$ *such that* $\mathcal{A}_n$ *recognizes* L_n *and* $\mathsf{Size}(\mathcal{A}_n)$ *is in* $\mathcal{O}(n^3.\log n)$.

Proof. As in some previous proofs, we will encode the value of a binary counter c with n clocks $(x_i)_{i=1..n}$. Roughly the encoding $b_1b_2\cdots b_n$ of counter c will be such that $b_i = 1$ whenever $x_i \geq 1$ and $b_i = 0$ whenever $x_i < 1$. Setting bit b_i to 0 is easy: we only need to reset clock x_i. Setting bit b_i to 1 needs a more involved construction: we use update $x_i :< 1$ which assigns non-deterministically to x_i a value less than 1, and we will check on the next transition that the value which has been assigned to x_i was close enough to 1 by checking $x_i \geq 1$. Automaton $\mathcal{A}_n$ then needs several states to remember what clock has just been updated (or equivalently what bit b_i has been updated to 1).

Automaton $\mathcal{A}_n$ will have $n+2$ states: an initial state, a final state, and one state for each carry which is propagated. State i is there to remember that bit b_i has just been set to 1 when the last a has happened. This mechanism is depicted below (we do not write other constraints for propagating carries, but this is similar to the construction done in previous proofs).



The complete automaton for L_3 is given in Example 13.

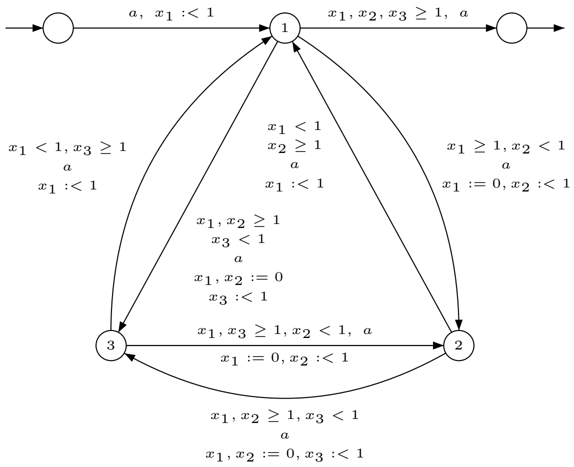
To prove correctness of the above construction, we define the application π which associates to each valuation $v \in \mathbb{T}^X$ a value of counter c (in set $\{0,\ldots,2^n-1\}$) as follows: the binary encoding of c is $b_1b_2\cdots b_n$ where $b_i = 1$ whenever $x_i \geq 1$ and $b_i = 0$ whenever $x_i < 1$. It is then easy to show the two properties below (we omit the proofs here):

(*i*) If $r = \langle q_0, v_0\rangle \xrightarrow[\tau_1]{a} \langle q_1, v_1\rangle \ldots \xrightarrow[\tau_p]{a} \langle q_p, v_p\rangle$ is an accepting run in $\mathcal{A}_n$, then for every $1 \leq i < p$, $\pi(v_i + \tau_{i+1} - \tau_i) = i$, and $p = 2^n$.

(*ii*) If $(\sigma,\tau) \in L_n$ and $\delta = \inf\{\tau_{i+1} - \tau_i \mid 1 \leq i < 2^n\}$, there is an accepting run for (σ,τ) in $\mathcal{A}_n$ which assigns the value $1-\delta$ to clock x_i when there is an update $x_i :< 1$ which is used.

Automaton $\mathcal{A}_n$ we have constructed has $n+2$ states, $n+2$ clocks (clocks $(x_i)_{1\leq i\leq n}$ above and two clocks y and z to ensure that the time sequence is strictly increasing and remains in the interval $]0,1[$), at most n^2+2 transitions, each transition has size at most $n.\log n$, the size of $\mathcal{A}_n$ is thus in $\mathcal{O}(n^3.\log n)$. This concludes the proof of Lemma 12. $\square$

Example 13 *The following automaton recognizes the language L_3. Implicitly, on each transition there is a guard $y > 0 \wedge z < 1$, and a reset $y := 0$ which enforces the time sequence to be strictly increasing and bounded by 1.*



Lemma 14 *Let $n \geq 0$. If $\mathcal{B}_n$ is a timed automaton of $\mathbf{Uta}_\varepsilon(\mathcal{C}_{df}, \mathcal{U}_0)$ which recognizes L_n, then $\mathsf{Size}(\mathcal{B}_n) \geq 2^n$.*

Proof. Note that we proved a close result (but in the case of timed automata without ε-transitions) in Lemma 4. We now show that it remains true even when ε-transitions are allowed.

Suppose that $\mathcal{B}_n$ has strictly less than 2^n states, and note m the smallest positive constant of $\mathcal{B}_n$. We consider a word $w = (a^{2^n}, \tau)$ such that $0 < \tau_1 < \cdots < \tau_{2^n} < m$. This word is accepted by $\mathcal{B}_n$: there exists an accepting path in $\mathcal{B}_n$ for w. We will show that we can remove a loop of this path, but we have to be careful when choosing the loop, as all loops cannot be removed (which was not the case when ε-transitions were not allowed). Consider an accepting run for w: before the first a and between two consecutive a's, some time has elapsed, so there is a state q_i where $\mathcal{B}_n$ waits a positive amount of time before taking the next (possibly ε) transition. Consider these 2^n states $(q_i)_{i=1..2^n}$, there exist $i < j$ such that $q_i = q_j$. We can remove the loop between q_i and q_j because the clock valuations before the i^{th}(resp. j^{th})-transition satisfy the following property: $\forall x \in X\ 0 < v(x) < m$; so the guards satisfied along the runs with and without the loop will be the same. $\square$

5. Conclusion

In this paper we have studied conciseness of several extensions of timed automata (using diagonal constraints, or updates), and we have proved that all these extensions, though decidable and not more expressive than classical timed automata, are exponentially more concise, which means that models that we can build using these features may be exponentially smaller than models based on classical timed automata.

An example of system which can be modelled in a natural way using these features are task scheduling problems, see for example [11]. In this case both diagonal constraints and updates are used, and an equivalent model using only non diagonal constraints and resets would be much bigger. Some of the updates (for example updates of the form $x := c$) and diagonal constraints can be used in UPPAAL [14], and we thus encourage people to use such features for modelling real systems.

References

[1] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[2] 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.

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

[4] 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.

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

[6] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2–3):291–345, 2004.

[7] Franck Cassez and Kim G. Larsen. The impressive power of stopwatches. In *Proc. 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.

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

[9] 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.

[10] 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.

[11] Elena Fersman, Paul Petterson, and Wang Yi. Timed automata with asynchrounous processes: Schedulability and decidability. In *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002.

[12] Thomas A. Henzinger. The theory of hybrid automata. In *Proc. 11th Annual Symposim on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.

[13] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Simone Tini. Concurrency in timed automata. *Theoretical Computer Science*, 309(1–3):503–527, 2003.

[14] 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.

[15] Nicolas Markey. Temporal logic with past is exponentially more succinct. *EATCS Bulletin*, 79:122–128, 2003.

[16] Nicolas Markey. Past is for free: On the complexity of verifying linear temporal properties with past. *Acta Informatica*, 40(6–7):431–458, 2004.