On the determinization of timed systems*

Patricia Bouyer[1], Samy Jaziri[1], Nicolas Markey[2]

[1] LSV – CNRS & ENS Paris-Saclay – France
[2] IRISA – CNRS & INRIA & Univ. Rennes 1 – France

Abstract. We introduce a new formalism called *automata over a timed domain* which provides an adequate framework for the determinization of timed systems. In this formalism, determinization w.r.t. timed language is always possible at the cost of changing the timed domain. We give a condition for determinizability of automata over a timed domain without changing the timed domain, which allows us to recover several known determinizable classes of timed systems, such as strongly-non-zeno timed automata, integer-reset timed automata, perturbed timed automata, etc. Moreover in the case of timed automata this condition encompasses most determinizability conditions from the literature.

1 Introduction

Timed automata. Timed automata [AD94] extend finite-state automata with real-valued variables, called *clocks*, that can be used to constrain delays between transitions along executions of an automaton. This is performed by decorating transitions with *timing constraints* and *clock resets*: timing constraints compare some of the clocks to integer values; a transition is then available only at times when the timing constraint is satisfied; clock resets set some of the clocks back to value zero.

Fig. 1 is a simple example representing the (simplified) behaviour of a computer mouse. Timed automata are very convenient to model real-time reactive systems: they enjoy polynomial-space analysis algorithms (with efficient implementations) for reachability (and many related verification problems), which is quite low in view of their expressiveness and handiness.

Determinization of timed automata. However, the situation is a bit less appealing in terms of the language-theoretic questions, where timed automata do not enjoy most of the nice properties of finite-state automata: they cannot be complemented nor determinized, and language inclusion and universality are undecidable [AD94]. As an example, the timed automaton of Fig. 2 does not admit a deterministic equivalent timed automaton: indeed, any deterministic[3] timed automaton accepting the same language would have to *guess* which occurrence of the letter `a` will have a matching `a` one time unit later. It can be proved that no finite timed automaton can achieve this. It has even been shown that

* This work was supported by ERC project EQualIS (308087).

[3] *Deterministic* for a timed automaton means that any two transitions out of the same state and carrying the same letter should have disjoint timing constraints.

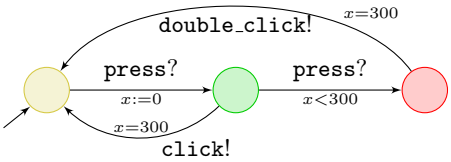

Fig. 1. Modelling a computer mouse

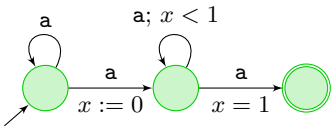

Fig. 2. A non-determinizable timed automaton

determinizability of a timed automaton is undecidable [Tri06,Fin06].

Two research directions have emerged from this situation. First, several subclasses of timed automata have been shown to allow determinization:

- *Event-clock automata* [AFH94] are automata in which each letter σ of the alphabet is associated with two clocks x_σ and y_σ: the former clock keeps track of the time elapsed since the last occurrence of σ in the execution, while the latter stores the time until the next occurrence of σ (if any). The class of event-clock automata has been proven to be closed under determinization [AFH94]. This result heavily relies on the fact that event-clock automata are *input-determined* [DT04], i.e., the values of the clocks (hence also the satisfaction of the guards) all along any execution of the automaton only depend on the input word; they do not depend on the execution itself.
- The class of timed automata with 0 as the only constant has been proven determinizable in [OW04], by determinizing its region automaton (and augmenting the resulting DFA with one clock to detect time elapses);
- *Integer-reset timed automata* [VPKM08] are timed automata in which clock resets may only occur at integer times (by constraining resetting transitions with $x = c$ for some clock x and integer c). The class of integer-reset timed automata is closed under determinization [VPKM08], by determinizing an enriched version of the region automaton and augmenting it with one clock.
- A common phenomenon in timed automata is that of time convergence: for instance, Zeno runs are infinite executions of timed automata along which the sum of the delays remains bounded. *Strongly-non-Zeno timed automata* [AMPS98] are timed automata in which any two entries in the same location are at least one time unit apart. It is proved in [BBBB09] that the class of strongly-non-Zeno timed automata is determinizable.
- *Perturbed* timed automata [ALM05] are timed automata whose semantics is perturbed by clock drifts, making clocks have different rates in $[1-\epsilon, 1+\epsilon]$ for some $0 < \epsilon < 1$. It is proved in [ALM05] that the *ϵ-perturbed language* of a timed automaton can be captured as the (non-perturbed) language of a deterministic timed automaton. This result is different in nature from the previous one, as it is not a closure property.

A second direction has focused on developing incomplete or approximation techniques for determinization. Several approaches have been proposed:

- In [BBBB09], the input timed automaton is unfolded into a tree with unboundedly many clocks; under some conditions, this tree may be refolded into a finite deterministic timed automaton. This technique can be used to

(re)prove several of the results listed above (event-clock automata, integer-reset timed automata, strongly-non-Zeno timed automata).
- Approximating techniques have also been developed: in [KT09], an algorithm is developed to compute a deterministic timed automaton, using a limited number of clocks, that *over-approximates* the language of the original timed automaton, by trying to keep track (as much as possible) of the states the input automaton can be in at each step.
- Finally, a game-based approach has been developed in [BSJK15]: it turns an automaton $\mathcal{A}$ into a two-player turn-based game, where winning strategies of the first player (with safety objective) can be turned into deterministic timed automata accepting the same language as $\mathcal{A}$. If the strategy is not winning, the resulting automaton would only over-approximate the language of $\mathcal{A}$.

Our contributions. In this paper, we consider a novel approach, based on a very expressive formalism for representing timed automata (and much more). Our formalism is based on *timed domains*, which are a versatile tool for representing the dynamics of continuous variables. Timed domains are equipped with update functions, corresponding to (but extending) clock resets of timed automata. Then automata over timed domains are automata built on these formalisms.

We propose various notions of determinism, and we discuss determinization procedures for some of them. We also discuss finite representation of those determinized automata. This new approach to the determinization of automata over timed domains allows to recover several existing results.

Related works. Besides the works already listed above, our approach was inspired by the approach of [BL12], even if the latter is not directly linked to determinization. To tackle the problem of minimization, the authors introduce a super-class of timed automata called *constrained timed register automata*, which is decidable and closed under minimization.

An extended version of this work will be available under the same title as an arXiV paper.

2 Definitions

2.1 Timed domains

Timed domains are our formalism for representing the evolution of continuous variables: a timed domain is made of *values* (e.g. vectors of nonnegative reals, which would correspond to clock valuations in timed automata) and a function encoding the evolution of those values when time elapses.

Definition 1. *A* timed domain *is a triple* $\mathcal{D} = \langle \mathcal{V}, \hookrightarrow \rangle$ *where* $\mathcal{V}$ *is a set of* values*, and* $\hookrightarrow \colon \mathcal{V} \times \mathbb{R}_{\geq 0} \to \mathcal{V}$ *is the time transition function, satisfying the condition* $\hookrightarrow(v, d + d') = \hookrightarrow(\hookrightarrow(v, d), d')$ *for all* $v \in \mathcal{V}$ *and all* $d, d' \in \mathbb{R}_{\geq 0}$.

In the sequel, we may write $v \overset{d}{\hookrightarrow} v'$ *for* $\hookrightarrow(v, d) = v'$.

Product and subset domains. Given two timed domains $\mathcal{D} = \langle \mathcal{V}, \hookrightarrow_{\mathcal{V}} \rangle$ and $\mathcal{D}' = \langle \mathcal{V}', \hookrightarrow_{\mathcal{V}'} \rangle$, we define their product $\mathcal{D} \times \mathcal{D}' = \langle \mathcal{V}_\times, \hookrightarrow_\times \rangle$ where $\mathcal{V}_\times = (\mathcal{V} \cup \{\bot\}) \times (\mathcal{V}' \cup \{\bot\})$ and $\hookrightarrow_\times((v, v'), d) = (\hookrightarrow_{\mathcal{V}}(v, d), \hookrightarrow_{\mathcal{V}'}(v', d))$ with $\hookrightarrow_{\mathcal{V}}(\bot, d) = \hookrightarrow_{\mathcal{V}'}(\bot, d) = \bot$. For $n \in \mathbb{N}_{>0}$, the timed domain $\mathcal{D}^n$ is defined inductively as $\mathcal{D}^1 = \mathcal{D}$ and $\mathcal{D}^{n+1} = \mathcal{D} \times \mathcal{D}^n$. Taking the product of timed domains can be viewed as considering multiple resources evolving synchronously. We add a special symbol $\bot$ to specify that a resource may be *inactive*; in that case, it does not evolve over time.

For a timed domain $\mathcal{D} = \langle \mathcal{V}, \hookrightarrow \rangle$, we also define its powerset $\mathcal{P}(\mathcal{D})$ as the timed domain $\langle \mathcal{V}_\mathcal{P}, \hookrightarrow_\mathcal{P} \rangle$ with $\mathcal{V}_\mathcal{P} = \mathcal{P}(\mathcal{V})$ and $\hookrightarrow_\mathcal{P}$ extends $\hookrightarrow$ to sets in the natural way.

Lemma 2. *Given timed domains $\mathcal{D}$ and $\mathcal{D}'$, and for any positive integer n, $\mathcal{D} \times \mathcal{D}'$, $\mathcal{D}^n$ and $\mathcal{P}(\mathcal{D})$ are timed domains.*

Example 1. Fix $M \in \mathbb{N}$. The M-bounded one-dimensional clock domain $\mathcal{D}_M = \langle \mathcal{C}_M, \hookrightarrow_{\mathcal{C}_M} \rangle$ is defined by $\mathcal{C}_M = [0; M] \cup \{+\infty\}$ equipped with the time transition function $\hookrightarrow_{\mathcal{C}_M}$ satisfying the requirements of the definition above, and such that $\hookrightarrow_{\mathcal{C}_M}(v, d) = v + d$ if $v + d \leq M$ and $\hookrightarrow(v, d) = \infty$ if $v + d > M$. The M-bounded n-dimensional clock domain is defined as the product $\mathcal{D}_M^n$, which we write $\mathcal{D}_M^n = \langle \mathcal{C}_M^n, \hookrightarrow_{\mathcal{C}_M^n} \rangle$. A value in the timed domain $\mathcal{D}_M^n$ corresponds to a clock valuation in timed automata [AD94]. Contrary to what is usually done, we explicitely replace every value larger than M with $+\infty$. As an example, $(0.3, 1.6, \bot) \in \mathcal{C}_2^3$ represents a clock valuation over three clocks, and $(0.3, 1.6, \bot) \overset{1.1}{\hookrightarrow}_{\mathcal{C}_2^3} (1.4, +\infty, \bot)$ represents a time-elapsing transition of 1.1 time units.

Example 2. Fix a continuous function $f \in \mathcal{C}^\infty(\mathbb{R}_{\geq 0}, \mathbb{R}^n \times \mathbb{R}^m)$ describing the evolution of two continuous variables x and y over time. We can define the timed domain $\mathcal{D}_f = \langle \mathbb{R}_{\geq 0} \times \mathbb{R}^n \times \mathbb{R}^m, \hookrightarrow_{\mathcal{D}_f} \rangle$ where $\hookrightarrow_{\mathcal{D}_f}$ satisfies the requirements of the definition above and such that $\hookrightarrow_{\mathcal{D}_f}((t, x, y), d) = (t + d, f(t + d))$. Such a timed domain would allow to define dynamical systems.

Many more examples of timed domains could be given, which would define rather complex systems evolving over time. For instance, we show in Section 4.3 how timed domains can be defined to represent *perturbed clocks*.

2.2 Updates

In this section, we introduce operations to be performed on values when taking transitions; it includes clock resets of timed automata, but is much more general. For $v \in \mathcal{V}$, we write $v^\mathcal{V} \colon \mathcal{V} \to \mathcal{V}$ for the constant function mapping all elements of $\mathcal{V}$ to v.

Definition 3. *Let $\mathcal{D} = \langle \mathcal{V}, \hookrightarrow \rangle$ be a timed domain, and Σ be a finite alphabet. An* update set *for $\mathcal{D}$ and Σ is a set $\Lambda \subseteq \Sigma \times \mathcal{V}^\mathcal{V}$.*

Given an update set Λ and a letter $\sigma \in \Sigma$, we write Λ_σ for the set $\{w \in \mathcal{V}^\mathcal{V} \mid (\sigma, w) \in \Lambda\}$. An element of Λ_σ is called a *σ-update*, or simply *update*.

Product update sets, subset update sets. Take a timed domain $\mathcal{D}$ equipped with an update set Λ over Σ. We equip $\mathcal{D}^n$ with its canonical update set, denoted Λ^n, and defined as follows:

$$\Lambda^n = \Big\{ (\sigma, (w_i \circ \pi^n_{k_i})_{1\leq i\leq n}) \;\Big|\; \sigma \in \Sigma \text{ and } \forall 1 \leq i \leq n.\ w_i \in \Lambda_\sigma \cup \{\bot^{\mathcal{V}}\} \text{ and } 1 \leq k_i \leq n \Big\}$$

where for $1 \leq b \leq a$, the function π^a_b is the projection $(d_j)_{1\leq j\leq a} \mapsto d_b$. Notice that we add to Λ_σ a function $\bot^{\mathcal{V}}$ which allows to set a "resource" inactive.

Given an update set Λ over $\mathcal{D} = \langle \mathcal{V}, \hookrightarrow \rangle$ and Σ, and given $p \in \mathbb{N}_{>0}$, we define an update set $\mathcal{P}^p(\Lambda)$ over $\mathcal{P}(\mathcal{D})^p$ and Σ as follows. Fix $\sigma \in \Sigma$ and $\boldsymbol{\gamma} = (\gamma_i)_{1\leq i\leq p}$ with $\gamma_i \subseteq \mathcal{V} \times \Lambda_\sigma$ for all $1 \leq i \leq p$. Each relation γ_i defines the possible updates of Λ_σ we can apply to a value $v \in \mathcal{V}$. To each γ_i we can associate a function $o_{\sigma,\gamma_i} \colon \mathcal{V} \to \mathcal{P}(\mathcal{V})$ which aggregates all possible updated values of v following instructions in γ_i: $o_{\sigma,\gamma_i}(v) = \{w(v) \mid (v, w) \in \gamma_i\}$. We extend o_σ, γ_i on $\mathcal{P}(\mathcal{V})$ and obtain $O_{\sigma,\gamma_i} \colon \mathcal{P}(\mathcal{V}) \to \mathcal{P}(\mathcal{V})$ which aggregates this time the possible updated values of all values in V (following instructions in γ_i): $O_{\sigma,\gamma_i}(V) = \bigcup_{v\in V} o_{\sigma,\gamma_i}(v)$. Finally, $O_{\sigma,\boldsymbol{\gamma}} \colon \mathcal{P}(\mathcal{V})^p \to \mathcal{P}(\mathcal{V})$ aggregates the possible updated values of $V_1, \ldots, V_p$, following respectively the instructions in $\gamma_1, \ldots, \gamma_p$: $O_{\sigma,\boldsymbol{\gamma}}((V_i)_{1\leq i\leq p}) = \bigcup_{1\leq i\leq p} O_{\sigma,\gamma_i}(V_i)$. In one line:

$$\begin{aligned} O_{\sigma,\boldsymbol{\gamma}} \colon \quad & \mathcal{P}(\mathcal{V})^p \to \mathcal{P}(\mathcal{V}) \\ & (V_i)_{1\leq i\leq p} \mapsto \bigcup_{1\leq i\leq p} \{w(\nu) \mid (\nu, w) \in \gamma_i \text{ and } \nu \in V_i\} \end{aligned}$$

Then $\mathcal{P}^p(\Lambda) =$ is the set

$$\Big\{ (\sigma, (O_{\sigma,\boldsymbol{\gamma}^j})_{1\leq j\leq p}) \;\Big|\; \sigma \in \Sigma \text{ and } \forall 1 \leq j \leq p,\ \boldsymbol{\gamma}^j \subseteq \mathcal{P}(\mathcal{V} \times \Lambda_\sigma)^p \Big\}.$$

From the remarks above, the resulting sets are indeed update sets:

Lemma 4. *If Λ and Λ' are update sets for $\mathcal{D}$ and $\mathcal{D}'$ over Σ, and if $n \in \mathbb{N}_{>0}$, then $\Lambda \times \Lambda'$ and Λ^n are a update sets respectively for $\mathcal{D} \times \mathcal{D}'$ and $\mathcal{D}^n$ over Σ.*

Example 3. Given Σ a finite alphabet, the one-dimensional clock domain $\mathcal{D}_M$ defined in Example 1 can be equipped with the (canonical) update set $\Lambda_M = \Sigma \times \{\mathtt{Id}, \mathtt{0}\}$, where $\mathtt{Id}(v) = v$ (that is, it keeps the clock value unchanged), and $\mathtt{0} = 0^{\mathcal{C}_M}$ (that is, it resets the clock to 0).

Then, $\mathcal{D}^n_M$ is equipped with operations of Λ^n_M (the product operations). Given an input vector $\mathbf{v} = (v_i)_{1\leq i\leq n} \in \mathcal{C}^n_M$, an operation ω of Λ^n_M is characterized by $\iota \colon \{1, \ldots, n\} \to \{1, \ldots, n\} \cup \{0, \bot\}$ such that for every $\mathbf{v}' = (v'_i)_{1\leq i\leq n}$, $\mathbf{v}' = \omega(\mathbf{v})$ if, and only, if:

$$v'_i = \begin{cases} 0 & \text{if } \iota(i) = 0 \\ \bot & \text{if } \iota(i) = \bot \\ v_j & \text{if } \iota(i) = j \end{cases}$$

Seeing $\mathbf{v}$ as a clock valuation, the i-th clock is reset in the first case, it is made inactive in the second case, and it takes the value of the j-th clock in the last case (note that if $j = i$, then the clock value is unchanged). We write ω_ι for the corresponding operation.

2.3 Automata over timed domains

Definition 5. *Fix a timed domain $\mathcal{D} = \langle \mathcal{V}, \hookrightarrow \rangle$ and an update set Λ for $\mathcal{D}$ over Σ. An* automaton *on $\mathcal{D}$ and Λ is a tuple $\mathcal{A} = \langle Q, q_{\mathsf{init}}, \nu_{\mathsf{init}}, T, F \rangle$ where Q is a finite set of states, $q_{\mathsf{init}} \in Q$ is an initial state, ν_{init} is an initial value, $T \subseteq Q \times \mathcal{V} \times \Lambda \times Q$ is the transition function, and $F \subseteq Q$ is the set of final states.*

Given an automaton $\mathcal{A}$ over $\mathcal{D}$ and Λ, we write $S_\mathcal{A}$ for the set $Q \times \mathcal{V}$ of *configurations* of $\mathcal{A}$. An automaton $\mathcal{A}$ induces a (possibly infinite) state transition system $\mathcal{S} = \langle S_\mathcal{A}, \to_\mathcal{A} \rangle$ where $\to_\mathcal{A} = (\xrightarrow{d}_\mathcal{A})_{d \in \mathbb{R}_{\geq 0}} \uplus (\xrightarrow{\sigma,w}_\mathcal{A})_{(\sigma,w) \in \Lambda}$, defined as follows:

$$\begin{aligned} (q, \nu) \xrightarrow{d}_\mathcal{A} (q', \nu') \quad &\Leftrightarrow \quad q = q' \text{ and } \nu \overset{d}{\hookrightarrow} \nu' \\ (q, \nu) \xrightarrow{\sigma,w}_\mathcal{A} (q', \nu') \quad &\Leftrightarrow \quad (q, \nu, (\sigma, w), q') \in T \text{ and } \nu' = w(\nu). \end{aligned}$$

Given a timed domain $\mathcal{D}$ and its update set Λ, and given $n \in \mathbb{N}_{>0}$, we write $\mathbb{A}_n(\mathcal{D}, \Lambda)$ for the set of all automata on $\mathcal{D}^n$ and Λ^n. Notice that $\mathbb{A}_n(\mathcal{D}, \Lambda) = \mathbb{A}_1(\mathcal{D}^n, \Lambda^n)$. We let $\mathbb{A}(\mathcal{D}, \Lambda) = \bigcup_{n \in \mathbb{N}_{>0}} \mathbb{A}_n(\mathcal{D}, \Lambda)$. Similarly, for $n \in \mathbb{N}$, we let $\mathcal{P}\mathbb{A}_n(\mathcal{D}, \Lambda) = \bigcup_{p \in \mathbb{N}} \mathbb{A}(\mathcal{P}(\mathcal{D}^n)^p, \mathcal{P}^p(\Lambda^n))$, and $\mathcal{P}\mathbb{A}(\mathcal{D}, \Lambda) = \bigcup_{n \in \mathbb{N}} \mathcal{P}\mathbb{A}_n(\mathcal{D}, \Lambda)$.

Remark 1. This definition of an automaton is half-way between standard automata and transition systems: there is no symbolic guards and symbolic guarded transitions, but a "list" of transitions, specifying, for each state, and for each value in the timed domain, what the next state should be, and how the value should be updated. This general form of automaton will be useful to apply a determinization procedure.

Example 4. An example of an automaton in $\mathbb{A}_1(\mathcal{D}_M, \Lambda_M)$ over $\Sigma = \{a\}$ is $\mathcal{A} = \langle \{q_1, q_2, q_3\}, q_1, 0, T, \{q_3\} \rangle$ where $T = \{(q_1, \nu, (a, \mathtt{0}^{\mathcal{C}_M}), q_2) \mid \nu \in \mathcal{C}_M\} \cup \{(q_2, \nu, (a, \mathtt{Id}), q_3) \mid \nu \in \mathcal{C}_M \cap \mathbb{Q}\}$. It generates (for instance) the sequences

$$(q_1, 0) \xrightarrow{d_1}_\mathcal{A} (q_1, d_1) \xrightarrow{a, \mathtt{0}^{\mathcal{C}_M}} (q_2, 0) \xrightarrow{d_2}_\mathcal{A} (q_2, d_2) \xrightarrow{a, \mathtt{Id}} (q_3, d_2)$$

requiring that d_2 is a rational number bounded by M.

2.4 Finite representation of automata over timed domains

With our definition, each transition $(q, \nu, (\sigma, w), q')$ is only available from configuration (q, ν). In general, the set of transitions is infinite. However, in order to get a finite representation, we may group transitions together.

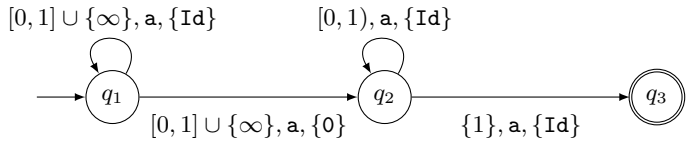

Fig. 3. A finitely-representable timed automaton $\mathcal{B}$

Let $\mathcal{G} \subseteq \mathcal{P}(\mathcal{V})$; we call it a *set of guards*. A $\mathcal{G}$*-guarded update* for $\sigma \in \Sigma$ is a pair $(G, O) \in \mathcal{G} \times \mathcal{P}(\Lambda_\sigma)$. A set $\{(G_i, O_i) \mid i \in I\}$ (I being a finite or infinite subset of $\mathbb{N}$) of $\mathcal{G}$-guarded updates for σ is *(i)* sound from q to q' whenever for every $i \in I$, for every $\nu \in G_i$, for every $w \in O_i$, $(q, \nu, (\sigma, w), q') \in T$; and *(ii)* complete from q to q' whenever for every $(q, \nu, (\sigma, w), q') \in T$, there exists $i \in I$ such that $\nu \in G_i$ and $w \in O_i$.

An automaton $\mathcal{A} = \langle Q, q_{\mathsf{init}}, \nu_{\mathsf{init}}, T, F \rangle$ is *finitely representable* using $\mathcal{G}$ whenever for every q and q' in Q, for every $\sigma \in \Sigma$, there exists a finite set of $\mathcal{G}$-guarded updates for σ, which is sound and complete from q to q'. In that case, there is a natural way to graphically represent the automaton, by depicting a transition for every $\mathcal{G}$-guarded update involved in the representation. We illustrate those representations in the following example.

Example 5. We consider the automaton $\mathcal{B} \in \mathbb{A}(\mathcal{D}_1, \Lambda_1)$ (over the one-dimensional clock domain) represented on Fig. 3, which as we explain corresponds to the timed automaton of Fig. 2. The guarded transition $q_1 \xrightarrow{[0,1]\cup\{\infty\},\mathtt{a},\{\mathtt{0}\}} q_2$ represents all the transitions $(q_1, \nu, (\mathtt{a}, \mathtt{0}), q_2)$ of $\mathcal{B}$, with $\nu \in [0,1] \cup \{\infty\} = \mathcal{C}_1 \setminus \{\bot\}$. This automaton has a single clock variable, and the above transition resets the variable to 0, whatever its original value. The guarded transition $q_2 \xrightarrow{\{1\},\mathtt{a},\{\mathtt{Id}\}} q_3$ checks that the value of the variable is 1 prior to going to q_3. Later we may write $[0, \infty)$ for $[0,1] \cup \{\infty\}$ when considering the one-dimensional clock domain $\mathcal{D}_1$.

Following this example, we remark that n-clocks timed automata with clock constraints bounded by M [AD94] correspond to those automata in $\mathbb{A}(\mathcal{C}_M^n, \Lambda_M^n)$, which can be finitely represented using guards of the form $(I_1, \ldots, I_n) \in \mathcal{I}_M^n$, where $\mathcal{I}_M$ is the set of intervals I whose bounds are nonnegative integral constants bounded by M, or $+\infty$. Strictly speaking, the current model allows transfers of clocks (using the updates $\omega_\iota \in \Lambda_M^n$ – see page 5), but we know that such updates can be expressed in timed automata [BDFP04]. In the following, we call *timed automata* those automata in the set:

$$\bigcup_{M \in \mathbb{N}} \bigcup_{n \in \mathbb{N}} \left\{ \mathcal{A} \in \mathbb{A}(\mathcal{C}_M^n, \Lambda_M^n) \;\middle|\; \begin{array}{l} \mathcal{A} \text{ can be finitely represented} \\ \text{using guards of the form } (I_1, \ldots, I_n) \in \mathcal{I}_M^n \end{array} \right\}$$

2.5 Commands

We now introduce the notion of *commands*, which we use to define different kinds of determinism.

Definition 6. *Let $\mathcal{D}$ be a timed domain and Λ be an update set. Let $\mathcal{A} \in \mathbb{A}(\mathcal{D}, \Lambda)$. Let Γ be a set (called* command alphabet*). Let $c \in \Gamma$. The c-command of $\mathcal{A}$ is a subset $\overset{c}{\twoheadrightarrow}_{\mathcal{A}} \subseteq S_{\mathcal{A}} \times S_{\mathcal{A}}$ s.t., writing $\rightarrow_{\mathcal{A}}^{+}$ for the transitive closure of $\rightarrow_{\mathcal{A}}$,*

$$(q,\nu) \overset{c}{\twoheadrightarrow}_{\mathcal{A}} (q',\nu') \implies (q,\nu) \rightarrow_{\mathcal{A}}^{+} (q',\nu')$$

A command *for a class $\mathbf{C}$ of automata over a timed domain $\mathcal{D}$ is a set $\kappa = (\overset{c}{\twoheadrightarrow})_{c\in\Gamma}$ where $\overset{c}{\twoheadrightarrow}$ maps each automaton $\mathcal{A}$ of $\mathbf{C}$ to a command $\overset{c}{\twoheadrightarrow}_{\mathcal{A}}$ of $\mathcal{A}$.*

Notice that some transitions from the automaton may be lost, and correspond to no command.

Fix a timed domain $\mathcal{D}$ and an update set Λ, a set $\mathbf{C}$ of automata over $\mathcal{D}$ and Λ, a command $\kappa = (\overset{c}{\twoheadrightarrow})_{c\in\Gamma}$ over $\mathbf{C}$. Let $\mathcal{A}$ be an automaton in $\mathbf{C}$. A κ-trace from a configuration (q,ν) is a finite sequence $\tau = (q_i,\nu_i)_{0\leq i\leq n}$ where $(q_0,\nu_0) = (q,\nu)$, and for which there exists a word $C = (c_i)_{1\leq i\leq n} \in \Gamma^n$ such that $(q_i,\nu_i) \overset{c_i}{\twoheadrightarrow} (q_{i+1},\nu_{i+1})$ for all $1 \leq i \leq n$. Trace τ is then said to be *generated* by C. Notice that a single word $C \in \Gamma^n$ may generate several traces (even from a single configuration), and that several words may generate the same trace. For a word $C \in \Gamma^n$, we write $\mathcal{T}_{\mathcal{A}}^{\kappa}((q,\nu),C)$ for the set of traces from (q,ν) generated by C.

Definition 7. *An automaton $\mathcal{A} \in \mathbf{C}$ is said κ-deterministic if, for any $C \in \Gamma^*$, the cardinality of $\mathcal{T}_{\mathcal{A}}^{\kappa}((q_{\mathsf{init}},\nu_{\mathsf{init}}),C)$ is at most* 1.

A word $C \in \Gamma^*$ reaches a configuration (q',ν') from (q,ν) w.r.t. κ if there exists a trace $\tau = (q_i,\nu_i)_{0\leq i\leq n} \in \mathcal{T}_{\mathcal{A}}^{\kappa}(C)$ from (q,ν) with $(q_n,\nu_n) = (q',\nu')$. Then (q',ν') is said *κ-reachable* from (q,ν); we write $S_{\mathcal{A}}^{\kappa}(q,\nu)$ for the set of κ-reachable configurations from (q,ν). For all the notations introduced above, we may omit to mention (q,ν) when we mean $(q_{\mathsf{init}},\nu_{\mathsf{init}})$.

Finally, a word $C \in \Gamma^n$ is accepted by $\mathcal{A}$ from (q,ν) if there is a trace $\tau \in \mathcal{T}_{\mathcal{A}}^{\kappa}((q,\nu),C)$ whose last configuration is in $F \times \mathcal{V}$. For a set of configurations $S \subseteq S_{\mathcal{A}}$, we write $\mathcal{L}_{\kappa}(\mathcal{A},S)$ for the set of words accepted by $\mathcal{A}$ from some $(q,\nu) \in S$. Finally, $\mathcal{L}_{\kappa}(\mathcal{A})$ corresponds to $\mathcal{L}_{\kappa}(\mathcal{A},\{q_{\mathsf{init}},\nu_{\mathsf{init}}\})$.

Proposition 8. *An automaton $\mathcal{A} \in \mathbf{C}$ is κ-deterministic if, and only if, for any $c \in \Gamma$ and any κ-reachable configuration (q,ν) of $\mathcal{A}$, there is at most one configuration (q',ν') such that $(q,\nu) \overset{c}{\twoheadrightarrow}_{\mathcal{A}} (q',\nu')$.*

2.6 Different notions of determinism

We consider two different types of commands, leading to two notions of accepted language and two notions of determinism that we study in the sequel.

Full command. The *full command* corresponds to $\Gamma_{\mathtt{F}} = \mathbb{R}_{\geq 0} \uplus \Lambda$: in this setting, a word contains full information about the operations that have been performed

on the values. More precisely, the *full command* of $\mathcal{A}$ over $\Gamma_{\mathtt{F}}$ is the relation $\twoheadrightarrow^{\mathtt{F}}_{\mathcal{A}}$ defined as

$$\begin{aligned}
(q,\nu)\overset{d}{\twoheadrightarrow}{}^{\mathtt{F}}_{\mathcal{A}}(q,\nu') &\;\Leftrightarrow\; \nu \overset{d}{\hookrightarrow} \nu' && \forall d \in \mathbb{R}_{\geq 0}\\
(q,\nu)\overset{a,w}{\longrightarrow\!\!\!\!\!\twoheadrightarrow}{}^{\mathtt{F}}_{\mathcal{A}}(q',\nu') &\;\Leftrightarrow\; (q,\nu,(a,w),q') \in T \text{ and } \nu' = w(\nu) && \forall (a,w) \in \Lambda.
\end{aligned}$$

Then $\kappa_{\mathtt{F}} = (\overset{c}{\twoheadrightarrow}{}^{\mathtt{F}})_{c\in\Gamma_{\mathtt{F}}}$ is the *full command* over $\mathbb{A}(\mathcal{D},\Lambda)$.

Being deterministic for the full command is not very demanding: it just amounts to satisfying that if $(q,\nu,(a,w),q_1) \in T$ and $(q,\nu,(a,w),q_2) \in T$, then $q_1 = q_2$. Thus the operator (of the commands) has access to all the variables of the system. This is the kind of determinism that is used e.g. for event-clock timed automata [AFH94]—we discuss this further in Section 4.2.

Timed command. The *timed command* corresponds to $\Gamma_{\mathtt{T}} = \mathbb{R}_{\geq 0} \uplus \Sigma$: this gives rise to the classical setting of *timed words*, with $\twoheadrightarrow^{\mathtt{T}}_{\mathcal{A}}$ defined as

$$\begin{aligned}
(q,\nu)\overset{d}{\twoheadrightarrow}{}^{\mathtt{T}}_{\mathcal{A}}(q,\nu') &\;\Leftrightarrow\; \nu \overset{d}{\hookrightarrow} \nu' && \forall d \in \mathbb{R}_{\geq 0}\\
(q,\nu)\overset{a}{\twoheadrightarrow}{}^{\mathtt{T}}_{\mathcal{A}}(q',\nu') &\;\Leftrightarrow\; (q,\nu,(a,w),q') \in T \text{ and } \nu' = w(\nu) && \forall (a,w) \in \Lambda.
\end{aligned}$$

Then $\kappa_{\mathtt{T}} = (\overset{c}{\twoheadrightarrow}{}^{\mathtt{T}})_{c\in\Gamma_{\mathtt{T}}}$ is the *timed command* over $\mathbb{A}(\mathcal{D},\Lambda)$.

This corresponds to the usual notion of determinism used for timed automata [AD94]. In a sense, the operator (of the commands) has access to the absolute time value (starting from value ν_0 at time 0) and to the action to be played.

Remark 2. We could define many other command sets, with the idea to finely describe which resources of the system the operator can access. Interesting command alphabets include partial observation (either of the variables or of the action alphabet of the system). For instance, following [DM02], consider a plant $\mathcal{P}$ given by a timed automaton with controllable (Σ_c) and uncontrollable (Σ_u) actions; an interesting command set would then be $\mathbb{R}_{\geq 0} \cup \Sigma_c$: the operator would then control delays and controllable actions, but could not control nor observe uncontrollable actions. Exploring such commands is part of our future work.

3 Determinization of ATD

3.1 Full-command determinization

In this section, we consider the full command $\kappa_{\mathtt{F}}$, over the alphabet $\Gamma_{\mathtt{F}} = \mathbb{R}_{\geq 0} \cup \Lambda$.

Theorem 9. *Let $\mathcal{D}$ be a timed domain and Λ be an update set. For any $\mathcal{A} \in \mathbb{A}_1(\mathcal{D},\Lambda)$, there exists a κ_F-deterministic automaton $\mathcal{A}_{det} \in \mathbb{A}_1(\mathcal{D},\Lambda)$ such that $\mathcal{L}_{\kappa_F}(\mathcal{A}) = \mathcal{L}_{\kappa_F}(\mathcal{A}_{det})$.*

Proof (Sketch). The proof follows the classical determinization procedure by powerset construction. We fix $\mathcal{A} = \langle Q, q_{\mathsf{init}}, \nu_{\mathsf{init}}, T, F\rangle \in \mathbb{A}_1(\mathcal{D}, \Lambda)$, and construct $\mathcal{A}_{\mathsf{det}} = \langle \mathcal{P}(Q), \{q_{\mathsf{init}}\}, \nu_{\mathsf{init}}, T_{\mathsf{det}}, F_{\mathsf{det}}\rangle$ with $F_{\mathsf{det}} = \{P \in \mathcal{P}(Q) \mid F \cap P \neq \emptyset\}$ and

$$(P, \nu, (a, w), P') \in T_{\mathsf{det}} \text{ iff } P' = \{q' \in Q' \mid \exists q \in P, (q, \nu, (a, w), q') \in T\}.$$

This automaton is in $\mathbb{A}_1(\mathcal{D}, \Lambda)$. $\kappa_{\mathtt{F}}$-determinism is straightforward from the definition of T_{det} and $\kappa_{\mathtt{F}}$. Finally, it is easily proven that $\mathcal{L}_{\kappa_{\mathtt{F}}}(\mathcal{A}) = \mathcal{L}_{\kappa_{\mathtt{F}}}(\mathcal{A}_{\mathsf{det}})$. □

Finite representation. If $\mathcal{A}$ can be finitely represented using guards in some set $\mathcal{G}$, then the automaton $\mathcal{A}_{\mathsf{det}}$ constructed in the previous can be (straightforwardly) finitely represented using boolean combinations of guards in $\mathcal{G}$.

3.2 Timed-command determinization

We recall that $\Gamma_{\mathtt{T}} = \mathbb{R}_{\geq 0} \uplus \Sigma$ is the timed-command alphabet. Determinizing with regards to that alphabet is the standard point-of-view used for timed systems.

3.2.1 General determinization. We first formalize a kind of powerset construction for our general automata. Note the change in the timed domain of the determinized automaton.

Theorem 10. *Let $\mathcal{D}$ be a timed domain and Λ be an update set. For any $\mathcal{A} \in \mathbb{A}_1(\mathcal{D}, \Lambda)$, there exists a κ_T-deterministic automaton $\mathcal{A}_{det} \in \mathcal{P}\mathbb{A}_1(\mathcal{D}, \Lambda)$ such that $\mathcal{L}_{\kappa_T}(\mathcal{A}) = \mathcal{L}_{\kappa_T}(\mathcal{A}_{det})$.*

Proof. Write $\mathcal{D} = \langle \mathcal{V}, \hookrightarrow\rangle$ and $\mathcal{A} = \langle Q, q_{\mathsf{init}}, \nu_{\mathsf{init}}, T, F\rangle$, and let Σ be the alphabet used by Λ. Write $Q = \{q_1, \ldots, q_p\}$, and assume w.l.o.g. that $q_{\mathsf{init}} = q_1$. For every $q \in Q$, define $\mathsf{ind}(q)$ the index of q, that is, i such that $q = q_i$.

We now construct a $\kappa_{\mathtt{T}}$-deterministic automaton in $\mathbb{A}_1(\mathcal{P}(\mathcal{D})^p, \mathcal{P}^p(\Lambda)) \subseteq \mathcal{P}\mathbb{A}_1(\mathcal{D}, \Lambda)$ accepting the same $\kappa_{\mathtt{T}}$-language as $\mathcal{A}$. For every $\mathbf{V} = (V_i)_{1\leq i\leq p} \in \mathcal{P}(\mathcal{V})^p$, we define the set $Q_{\mathbf{V}} = \{q \in Q \mid V_{\mathsf{ind}(q)} \neq \emptyset\}$. Intuitively, each set V_i represents the set of possible values at state q_i, hence $Q_{\mathbf{V}}$ represents the set of states the system can be in, when the possible values in each state is given by $\mathbf{V}$.

Fix $\sigma \in \Sigma$. For every $1 \leq i, j \leq p$, let $\gamma_\sigma^{i\to j}$ be the set $\{(\nu, w) \in \mathcal{V} \times \Lambda_\sigma \mid (q_i, \nu, (\sigma, w), q_j) \in T\}$ and $\boldsymbol{\gamma}_\sigma^{\to j} = (\gamma_\sigma^{i\to j})_{1\leq i\leq p}$. We define the following operation, which belongs to $\mathcal{P}^p(\Lambda)$ (see page 5):

$$O_{\sigma,\mathcal{A}} = (O_{\sigma,\boldsymbol{\gamma}_\sigma^{\to 1}}, \ldots, O_{\sigma,\boldsymbol{\gamma}_\sigma^{\to p}})$$

Somehow, $\gamma_\sigma^{i\to j}$ records how one can reach state q_j from state q_i with letter σ, i.e. the set of possible values, together with the set of updated values; and the operation $O_{\sigma,\boldsymbol{\gamma}_\sigma^{\to j}}(\mathbf{V})$ aggregates all the possible ways to reach q_j, if we start from some (q_i, ν) with $\nu \in V_i$ (with $1 \leq i \leq p$).

Lemma 11. *Let $\mathbf{V} = (V_i)_{1\leq i\leq p} \in \mathcal{P}(\mathcal{V})^p$, and $\mathbf{V}' = (V'_i)_{1\leq i\leq p} = O_{\sigma,\mathcal{A}}(\mathbf{V})$. Then, for every $q' \in Q$, for every $\nu' \in \mathcal{V}$:*

$$\nu' \in V'_{\mathsf{ind}(q')} \iff \exists (q, \nu, (\sigma, w), q') \in T \text{ s.t. } \nu \in V_{\mathsf{ind}(q)} \text{ and } \nu' = w(\nu)$$

We then let $\mathcal{A}_{\mathsf{det}} = \langle \mathcal{P}(Q), \{q_{\mathsf{init}}\}, (\{\nu_{\mathsf{init}}\}, \emptyset, \ldots, \emptyset), T_{\mathsf{det}}, F_{\mathsf{det}} \rangle$ where:

- T_{det} is made of the transitions $(Q_{\mathbf{V}}, \mathbf{V}, (\sigma, O_{\sigma,\mathcal{A}}), Q')$ where:
  - $\mathbf{V} \in \mathcal{P}(\mathcal{V})^p$
  - $Q' = Q_{\mathbf{V}'}$ where $\mathbf{V}' = O_{\sigma,\mathcal{A}}(\mathbf{V})$
- $F_{\mathsf{det}} = \{Q' \subseteq Q \mid Q' \cap F \neq \emptyset\}$.

Proposition 12. *$\mathcal{A}_{det}$ is κ_T-deterministic and $\mathcal{L}_{\kappa_T}(\mathcal{A}) = \mathcal{L}_{\kappa_T}(\mathcal{A}_{det})$.*

Proof (Sketch). The κ_{T}-determinism of $\mathcal{A}_{\mathsf{det}}$ is obvious (by Prop. 8) since, for every $\sigma \in \Sigma$, there is a unique operation associated with σ, namely $O_{\sigma,\mathcal{A}}$. It remains to show the equality of the two languages. We first define a correspondence between vectors $\mathbf{V}$ and sets of configurations of $\mathcal{A}$ as follows:

$$\begin{aligned} \phi\colon\ \mathcal{P}(\mathcal{V})^p &\to \mathcal{P}(S^{\mathcal{A}}) \\ \mathbf{V} &\mapsto \{(q,\nu) \mid q \in Q_{\mathbf{V}} \text{ and } \nu \in V_{\mathsf{ind}(q)}\} \end{aligned}$$

It is easy to see that this is a bijection. By induction, we can prove that for every $\mathbf{V} \in \mathcal{P}(\mathcal{V})^p$, $\mathcal{L}_{\kappa_{\mathrm{T}}}(\mathcal{A}_{\mathsf{det}}, (Q_{\mathbf{V}}, \mathbf{V})) = \mathcal{L}_{\kappa_{\mathrm{T}}}(\mathcal{A}, \phi(\mathbf{V}))$. □

Finite representation. We discuss now the finite representability of automaton $\mathcal{A}_{\mathsf{det}}$ constructed in the previous proof. We only consider the case of timed automata here, and have a more general discussion in the corresponding research report. The operations allowed in timed automata are ω_ι, where $\iota\colon \{1,\ldots,n\} \to \{1,\ldots,n\} \cup \{0,\bot\}$ (see page 5). We assume that in $\mathcal{A}$, there is a guard $G_{q,q',\sigma,\iota}$ defined with disjunctions and conjunctions (involving several clocks) of intervals constraints such that $(G_{q,q',\sigma,\iota}, \omega_\iota)$ is a guarded update from q to q' for σ.

The transitions between two states Q_1 and Q_2 of $\mathcal{A}_{\mathsf{det}}$ labelled by σ can then be written as:

- a constraint requiring that $\forall q_j \in Q_2$, $\exists q_i \in Q_1$, $V_i \cap \left(\bigcup_{\iota(j)\neq\bot} G_{q_i,q_j,\sigma,\iota}\right) \neq \emptyset$;
- a constraint requiring that $\forall q_j \notin Q_2$, $\forall q_i \in Q_1$, $V_i \cap \left(\bigcup_{\iota(j)\neq\bot} G_{q_i,q_j,\sigma,\iota}\right) = \emptyset$;
- for each $q_i \in Q_1$, for each $q_j \in Q_2$, for every ι, there are rules $V_i \overset{G_{q_i,q_j,\sigma,\iota},\omega_\iota}{\longmapsto} V'_j$, representing a transfer of valuations from V_i (for those valuations of V_i which belong to $G_{q_i,q_j,\sigma,\iota}$) to V'_j, after update ω_ι.

Example 6. Consider again the automaton $\mathcal{B}$ depicted on Fig. 3. The κ_{T}-deterministic automaton $\mathcal{B}_{\mathsf{det}}$ is depicted on Fig. 4, with the convention we have just discussed. Given that there are three states in $\mathcal{B}$ and one clock, the timed domain of $\mathcal{B}_{\mathsf{det}}$ is $\mathcal{P}(\mathcal{D}_1)^3$; hence there are three sets of clocks, one for each state of $\mathcal{B}$. We write V_1 (resp. V_2, V_3) for the set of clocks corresponding to q_1 (resp. q_2, q_3). As explained before, the guarded updates are represented explicitly as follows: we write $V_i \overset{G,O}{\longmapsto} V_j$ for "for each element $\nu \in V_i \cap G$, for each $w \in O$, add $w(\nu)$ to V_j". So, for instance, the transition between $\{q_1, q_2\}$ and $\{q_1, q_2, q_3\}$ is guarded by the existence of some $\nu \in V_2$ such that $\nu = 1$, and if this holds, then we perform action a and take the transition while keeping all values in V_1, adding a 0 to V_2, keeping all values but value 1 in V_2, and initializing V_3 with 1.

We realize that, while this is not really a timed automaton (since it involves unboundedly many clocks), all these clocks can be partitioned, and can be manipulated using first-order quantifications.

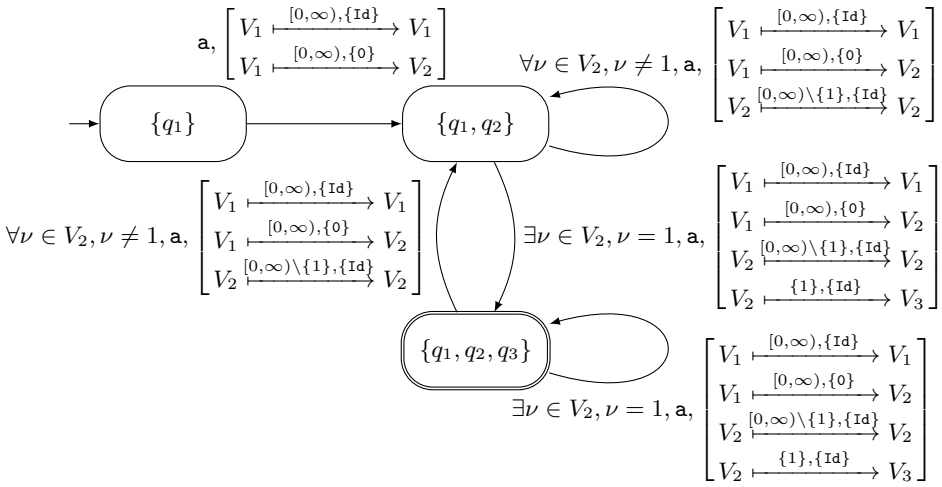

Fig. 4. Determinization of $\mathcal{B}$ of Figure 3

3.2.2 Strong determinization. We now focus on the case where the previously constructed deterministic automaton satisfies some nice boundedness property, which allows to *flatten* it (that is, if the original automaton is in $\mathbb{A}(\mathcal{D}^n, \Lambda^n)$, then so will be the determinized automaton).

We fix $\mathcal{A} = \langle Q, q_{\mathsf{init}}, \nu_{\mathsf{init}}, T, F \rangle \in \mathbb{A}_1(\mathcal{D}^n, \Lambda^n)$, and write $p = |Q|$. Borrowing notations from the proof of Theorem 10, every reachable state in $\mathcal{A}_{\mathsf{det}}$ is characterized by some $\mathbf{V} \in \mathcal{P}(\mathcal{V}^n)^p$. Furthermore, for such a vector $\mathbf{V}$, we write V_i for its i-th component (for every $1 \leq i \leq p$), and we use this implicit convention for all the vectors we manipulate; we also extend operations componentwise.

We say that $\mathcal{A}_{\mathsf{det}}$ is *m-weakly monotonic* whenever there exists $\mathbf{M} \in \mathbb{N}^p$, with $m = \sum_{i=1}^{p} M_i$, such that for every $\mathbf{V} \in \mathcal{P}(\mathcal{V}^n)^p$ in $\mathcal{A}_{\mathsf{det}}$, there exists $\mathbf{V}' \in \mathcal{P}(\mathcal{V}^n)^p$ such that (i) $Q_{\mathbf{V}} = Q_{\mathbf{V}'}$, (ii) $\mathbf{V}' \subseteq \mathbf{V}$, (iii) $|\mathbf{V}'| \leq \mathbf{M}$ (that is,$|V_i'| \leq M_i$ for every $1 \leq i \leq p$) and (iv) $\mathcal{L}_{\kappa_\mathrm{T}}(\mathcal{A}_{\mathsf{det}}, \mathbf{V}') = \mathcal{L}_{\kappa_\mathrm{T}}(\mathcal{A}_{\mathsf{det}}, \mathbf{V})$. The intuition behind this condition is that $\mathbf{V}'$ selects a bounded number of values out of $\mathbf{V}$, which are enough to pursue the computation correctly (reading and accepting only relevant words). Condition (i) is for ensuring one should stay in the same discrete state of the automaton for pursuing the computation; Condition (ii) ensures that one can keep the same kinds of updates (we preserve the set of values on which we can apply the updates); Condition (iii) bounds the size of the sets of selected values; Finally, condition (iv) ensures the correctness of $\mathbf{V}'$ w.r.t. $\mathbf{V}$.

Theorem 13. *Let $\mathcal{D}$ be a timed domain and Λ be an action domain. Let $\mathcal{A} \in \mathbb{A}_n(\mathcal{D}, \Lambda) = \mathbb{A}_1(\mathcal{D}^n, \Lambda^n)$, and write $\mathcal{A}_{det}$ for the automaton constructed in the proof of Theorem 10. Assume furthermore that there exists $m \in \mathbb{N}$ such that $\mathcal{A}_{det}$ is m-weakly monotonic. Then, there exists a κ_T-deterministic automaton $\mathcal{A}_{sdet} \in \mathbb{A}_1(\mathcal{D}^{nm}, \Lambda^{nm})$ such that $\mathcal{L}_{\kappa_T}(\mathcal{A}) = \mathcal{L}_{\kappa_T}(\mathcal{A}_{sdet})$.*

The idea is to represent each vector $\mathbf{V}$ such that $|\mathbf{V}| \leq \mathbf{M}$ by a single huge vector $\nu \in (\mathcal{V}^n)^m$ such that the first M_1 components of ν stores the elements of V_1, the next M_2 components stores the elements of V_2, etc. The element $\bot^n$ is

used to fill the components of ν which are not used by some element of V_i (this can happen when the cardinal of V_i is (strictly) smaller than M_i). Through this correspondence, we transform the transitions of $\mathcal{A}_{\mathsf{det}}$ into transitions over $(\mathcal{V}^n)^m$. In particular, to compute an update for some $\sigma \in \Sigma$ on ν, which corresponds to some $\mathbf{V}$, we apply $O_{\sigma,\mathcal{A}}$ on $\mathbf{V}$, reduce it using the condition given m-weak monotonicity, and reorder the resulting "small" vector of $\mathcal{P}(\mathcal{V}^n)^p$ into a vector $\nu' \in (\mathcal{V}^n)^m$.

Finite representation. It is not possible to obtain a finite representation for $\mathcal{A}_{\mathsf{sdet}}$ in general, even if $\mathcal{A}_{\mathsf{det}}$ can be finitely represented; indeed, the construction relies on a choice of $\mathbf{V}' \subseteq \mathbf{V}$, which is a priori arbitrary. This can however be used when all reachable $\mathbf{V}$ are such that $|\mathbf{V}| \leq \mathbf{M}$.

Actually, we can modify the m-weak monotonicity assumption of Theorem 13 into a more complex and abstract condition, so that the obtained deterministic automaton has a finite representation as soon as $\mathcal{A}_{\mathsf{det}}$ has a finite representation.

4 Applications

4.1 Applications to plain timed automata

We have already explained how Theorem 10 applies to timed automata, yielding deterministic automata $\mathcal{A}_{\mathsf{det}}$ in $\mathcal{P}\mathbb{A}(\mathcal{D}_M, \Lambda_M)$. Theorem 13 can be used to get a deterministic automaton in $\mathbb{A}(\mathcal{D}_M, \Lambda_M)$: our notion of m-weak monotonicity in a sense corresponds to the clock-boundedness condition of [BBBB09].

Our approach is actually a bit stronger, as it can capture other classes of determinizable timed automata, such as the class of finally-imprecise timed automata: a location q is imprecise if any word accepted from some configuration (q, ν) is also accepted from any other configuration (q, ν') in the same region; a timed automaton is then said finally-imprecise if after a fixed number m of discrete steps, it only visits imprecise states. We can prove that finally-imprecise timed automata do have an equivalent deterministic timed automata. This class actually encompasses all timed automata with 0 as the only constant [OW04].

4.2 Applications to event-clock automata

In order to capture event-clock automata [AFH94] in our formalism, we first define the *event-clock domain*. We fix a maximal constant M, and let $\mathcal{E}_M = ([0, M] \cup \{+\infty, \bot\})^2$: the first component corresponds to an event-recording clock, while the second is for event-predicting clocks. For $d \in \mathbb{R}_{\geq 0}$, we then set $(x, y) \overset{d}{\hookrightarrow}_{\mathcal{E}_M} (x', y')$ whenever $x \overset{d}{\hookrightarrow}_{\mathcal{C}_M} x'$, and $y' = y - d$ if $y - d \geq 0$, and $y' = +\infty$ otherwise (and $y' = \bot$ if, and only if, $y = \bot$). Thus the first component corresponds to the M-bounded one-dimensional clock domain defined at Example 1 (with an additional symbol $\bot$ when the clock is inactive). This defines the M-bounded one-letter event-clock domain $\mathcal{F}_M = \langle \mathcal{E}_M, \hookrightarrow_{\mathcal{E}_M} \rangle$. Given an alphabet $\Sigma = \{\sigma_i \mid 1 \leq i \leq n\} \uplus \{\mathsf{init}\}$ (see below), the M-bounded Σ-event-clock domain, denoted $\mathcal{F}_M^\Sigma$, is the timed domain $\mathcal{F}_M^n$.

We now associate updates with this timed domain; for this we reuse the projections π_b^a we defined in Section 2.2: we define the action domain Θ_M on $\mathcal{E}_M$ as $\{(\sigma, (w \circ \pi_1^2, w' \circ \pi_2^2)) \mid \sigma \in \Sigma,\ w \in \{\mathtt{Id}, \mathtt{0}, \bot\},\ w' \in \{\mathtt{Id}, \bot\} \cup \{\mathbf{d} \mid d \in \mathbb{R}_{\geq 0}\}\}$. Again, we extend this action domain to $\mathcal{F}_M^\Sigma$, denoting the resulting action domain with Θ_M^Σ.

We now define M-bounded Σ-event-clock automata. For this, we set $\Sigma = \{\sigma_i \mid 1 \leq i \leq n\} \cup \{\mathsf{init}\}$, where init is a special symbol used only for initializing the automaton. An automaton $\mathcal{A} = \langle Q, q_{\mathsf{init}}, \nu_{\mathsf{init}}, T, F\rangle$ is in the class $\mathsf{ECA}(\Sigma, M)$ of M-bounded Σ-event-clock automata if

- $\nu_{\mathsf{init}} = (\bot)_{1 \leq i \leq 2n}$, and $(q_{\mathsf{init}}, \nu_{\mathsf{init}})$ only initializes the computation, with transitions $(q_{\mathsf{init}}, \nu_{\mathsf{init}}, (\mathsf{init}, (\bot, d_i)_{1 \leq i \leq n}), q_1)$ for each $(d_i)_{1 \leq i \leq n} \in (\mathbb{R}_{\geq 0} \cup \{\bot\})^n$;
- for any transition $(q, \nu, (\sigma_i, w), q') \in T$ with $\sigma \neq \mathsf{init}$ and $q \neq q_{\mathsf{init}}$, variable y_i must have value 0 in ν, and operation w must set variable x_i to 0, variable y_i to some value in $[0, M] \cup \{\bot\}$, and leave the other variables unchanged.
- finally, for any transition $(q, \nu, (\sigma_i, w), q_f)$ with $q_f \in F$, we require that $y_j = \bot$ in ν for any $j \neq i$, and that $y_i = 0$.

An important feature of event-clock automata is that they are *input-determined*: in our setting, this can be expressed as an isomorphism between $\mathcal{L}_{\kappa_\mathtt{T}}(\mathcal{A})$ and $\mathcal{L}_{\kappa_\mathtt{F}}(\mathcal{A})$: intuitively, the operations performed on the clocks can be derived from observing the time of occurrence of the letters along words. Now, applying Theorem 9 to an event-clock automaton $\mathcal{A}$, we get a $\kappa_\mathtt{F}$-deterministic automaton $\mathcal{A}_{\mathsf{det}}$ accepting the same $\kappa_\mathtt{F}$-language as $\mathcal{A}$, hence also the same language (in the usual sense). Moreover, $\mathcal{A}_{\mathsf{det}}$ is easily proved to lie in $\mathsf{ECA}(\Sigma, M)$.

Remark 3. The automaton $\mathcal{A}_{\mathsf{det}}$ is *not* $\kappa_\mathtt{T}$-deterministic, since from any configuration, there are several transitions, each having a different "guess" for updating the clock y_i associated with the letter carried by the transition. This is also the case of the determinization result of [AFH94].

If $\mathcal{A}$ only involves *event-recording* clocks, then so does $\mathcal{A}_{\mathsf{det}}$. Thus the resulting automaton does not have to guess values for clocks y_i, and it is $\kappa_\mathtt{T}$-deterministic.

4.3 Application to perturbed timed automata

The model of *perturbed timed automata* has been proposed in [ALM05], with the idea that adding perturbations to the system can indeed help having interesting properties like determinizability. The syntax of this model is a standard timed automaton, but its semantics is parametrized by some $\epsilon \in (0, 1)$: in this model, we consider that the slope of a clock can be perturbed by at most ϵ. It is shown in [ALM05] that single-clock perturbed timed automata can be determinized into standard timed automata. We can fit this model into our framework.

To track the possible slopes of a clock, we use two "variables", one which runs at speed $1 - \epsilon$, and the other at speed $1 + \epsilon$. If $M \in \mathbb{N}$, the M-bounded one-dimensional ϵ-perturbed clock domain is $\mathcal{D}_{M,\epsilon} = \langle \mathcal{C}_{M,\epsilon}, \hookrightarrow_{\mathcal{C}_{M,\epsilon}}\rangle$ with:

- $\mathcal{C}_{M,\epsilon} = ([0, M(1+\epsilon)] \cup \{\infty\})^2$;

– $(x^-, x^+) \overset{d}{\hookrightarrow}_{\mathcal{C}_{M,\epsilon}} (x^- + d(1-\epsilon), x^+ + d(1+\epsilon))$ with conventions similar to the clock domain for manipulating ∞.

The two values x^- and x^+ represent respectively the lowest and greatest value that the perturbed clock x can take.

We equip this one-dimensional perturbed clock domain with a subset of the canonical action domain on two clocks, where updates on x^- and x^+ are forced to be the same. It is then easy to define a set of guards $\mathcal{I}_{M,\epsilon}$ such that any one-clock perturbed timed automata of [ALM05] can be represented by an automaton in $\mathbb{A}_1(\mathcal{D}_{M,\epsilon}, \Lambda_{M,\epsilon})$, which is finitely representable using guards in $\mathcal{I}_{M,\epsilon}$.

We can show that a proof very similar to that of Theorem 13 (or to its modified version) can be used to determinize the automaton. The result is not a timed automaton, but can be modified into a real timed automaton. This allows to recover the determinizability result of [ALM05].

5 Conclusions and future work

In this work, we have proposed a general model of automata based on a timed domain, and general notions of updates over that domain. We have discussed the notion of determinism for this model, by defining the notion of commands and discussing some possible sets of such commands. For two of these sets of commands (the full command, and the timed command), we have designed a generic procedure for determinizing the automata. While the full-command determinization stays within the class of automata we start with, the timed-command determinization involves a powerset construction, which increases the number of "variables" the automaton can manipulate. We have exhibited conditions under which this construction can be flattened into the original class of automata. We have applied our approach mostly to timed-automata-like classes of systems, and recovered many existing determinizability results. In particular, our approach gives a good understanding of event-clock timed automata [AFH94], gives a fresh view over the generic unfolding procedure for standard timed automaton of [BBBB09], and allows to recover the determinizability result for single-clock perturbed timed automata [ALM05].

As illustrated all along the paper, our framework encompasses timed automata and can represent various kinds of dynamical systems, but also timed systems with richer discrete structures. We can e.g. fit into our framework some families of pushdown timed automata, by encoding in the timed domain the "clock values" possibly stored in a stack. While it is not clear yet whether our approach can yield new results for those systems, we believe it is worth investigating.

Further, we believe that the notion of commands and the different kinds of determinism it generates are interesting. As illustrated in Example 2, we believe this approach could be worth investigating for monitoring or controller synthesis.

Finally, it is not completely clear to us how our approach for timed automata compares to the game approach of [BSJK15], so this would be worth investigating as well. Also, the fact that perturbations can be encoded in the timed domain (recall Section 4.3) might also have some interest for robustness issues.

References

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

[AFH94] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. In CAV'94, LNCS 818, p. 1–13. Springer, 1994.

[ALM05] R. Alur, S. La Torre, and P. Madhusudan. Perturbed timed automata. In HSCC'05, LNCS 3414, p. 70–85. Springer, 2005.

[AMPS98] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In SSC'98, p. 469–474. Elsevier, 1998.

[BBBB09] Ch. Baier, N. Bertrand, P. Bouyer, and Th. Brihaye. When are timed automata determinizable? In ICALP'09, LNCS 5556, p. 43–54. Springer, 2009.

[BDFP04] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.

[BL12] M. Bojańczyk and S. Lasota. A machine-independent characterization of timed languages. In ICALP'12, LNCS 7392, p. 92–103. Springer, 2012.

[BSJK15] N. Bertrand, A. Stainer, T. Jéron, and M. Krichen. A game approach to determinize timed automata. *Formal Methods in System Design*, 46(1):42–80, 2015.

[DM02] D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In STACS'02, LNCS 2285, p. 571–582. Springer, 2002.

[DT04] D. D'Souza and N. Tabareau. On timed automata with input-determined guards. In FORMATS-FTRTFT'04, LNCS 3253, p. 68–83. Springer, 2004.

[Fin06] O. Finkel. Undecidable problems about timed automata. In FORMATS'06, LNCS 4202, p. 187–199. Springer, 2006.

[KT09] M. Krichen and S. Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3):238–304, 2009.

[OW04] J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In LICS'04, p. 54–63. IEEE Comp. Soc. Press, 2004.

[Tri06] S. Tripakis. Folk theorems on the determinization and minimization of timed automata. *Information Processing Letters*, 99(6):222–226, 2006.

[VPKM08] P. Vijay Suman, P. K. Pandya, S. N. Krishna, and L. Manasa. Timed automata with integer resets: Language inclusion and expressiveness. In FORMATS'08, LNCS 5215, p. 78–92. Springer, 2008.