Event Clock Automata: from Theory to Practice*

G. Geeraerts[1]**, N. Sznajder[2], and J.F. Raskin[1]

[1] Université Libre Bruxelles, Département d'Informatique, Brussels, Belgium
[2] Université Pierre et Marie Curie, UMR CNRS 7606, LIP6, Paris, France.
{gigeerae,jraskin}@ulb.ac.be, nathalie.sznajder@lip6.fr

Abstract. *Event clock automata* (ECA) are a model for *timed languages* that has been introduced by Alur, Fix and Henzinger as an alternative to *timed automata*, with better theoretical properties (for instance, ECA are determinizable while timed automata are not). In this paper, we *revisit* and *extend* the theory of ECA. We first prove that *no finite time abstract language equivalence* exists for ECA, thereby disproving a claim in the original work on ECA. This means in particular that regions *do not form a time abstract bisimulation*. Nevertheless, we show that regions can still be used to build a finite automaton recognizing the *untimed language of an* ECA. Then, we extend the classical notions of *zones* and *DBMs* to let them handle event clocks instead of plain clocks (as in timed automata) by introducing *event zones* and *Event DBMs* (EDBMs). We discuss algorithms to handle event zones represented as EDBMs, as well as (semi-) algorithms based on EDBMs to decide language emptiness of ECA.

1 Introduction

Timed automata have been introduced by Alur and Dill in the early nineties [2] and are a successful and popular model to reason about *timed behaviors* of computer systems. Where finite automata represent behaviors by finite sequences of actions, timed automata define sets of *timed words* (called *timed languages*) that are finite sequences of actions, each paired with a real time stamp. To this end, timed automata extend finite automata with a finite set of real valued clocks, that can be tested and reset with each action of the system. The theory of timed automata is now well developed [1]. The algorithms to analyse timed automata have been implemented in several tools such as Kronos [7] or UppAal (which is increasingly applied in industrial case studies) [4].

Timed automata, however, suffer from certain weaknesses, at least from the theoretical point of view. As a matter of fact, timed automata are *not determinizable* and *cannot be complemented* in general [2]. Intuitively, this stems from the fact that the reset of the clocks cannot be made deterministic wrt the word being read. Indeed, from a

* Work supported by the projects: (i) QUASIMODO (FP7- ICT-STREP-214755), Quasimodo: "Quantitative System Properties in Model-Driven-Design of Embedded", http://www.quasimodo.aau.dk/, (ii) GASICS (ESF-EUROCORES LogiCCC), Gasics: "Games for Analysis and Synthesis of Interactive Computational Systems", http://www.ulb.ac.be/di/gasics/ and (iii) Moves: "Fundamental Issues in Modelling, Verification and Evolution of Software", http://moves.ulb.ac.be, a PAI program funded by the Federal Belgian Government.

** Partly supported by a 'Crédit aux chercheurs' from the Belgian FRS/F.N.R.S.

given location, there can be two transitions, labeled by the same action a but different reset sets.

This observation has prompted Alur, Fix and Henzinger to introduce the class of *event clock automata* (ECA for short) [3], as an alternative model for timed languages. Unlike timed automata, ECA force the clock resets to be strongly linked to the occurrences of actions. More precisely, for each action a of the system, there are two clocks $\overleftarrow{x_a}$ and $\overrightarrow{x_a}$ in an ECA: $\overleftarrow{x_a}$ is the *history clock* of a and *always records the time elapsed since the last occurrence of* a. Symmetrically, $\overrightarrow{x_a}$ is the *prophecy clock* for a, and *always predicts the time distance up to the next occurrence of* a. As a consequence, while history clocks see their values *increase* with time elapsing (like clocks in timed automata do), the values of prophecy clocks *decrease over time*. However, this scheme ensures that the value of any clock is uniquely determined at any point in the timed word being read, no matter what path is being followed in the ECA. A nice consequence of this definition is that ECA are *determinizable* [3]. While the theory of ECA has witnessed some developments [12, 10, 14, 8, 11] since the seminal paper, no tool is available that exploits the full power of event clocks (the only tool we are aware of is TEMPO [13] and it is restricted to *event-recording automata*, i.e. ECA with history clocks only).

In this paper, we revisit and extend the theory of ECA, with the hope to make it more practical and amenable to implementation. A widespread belief [3] about ECA and their analysis is that ECA are similar enough to timed automata that the classical techniques (such as regions, zones or DBMs) developed for them can readily be applied to ECA. The present research, however, highlights *fundamental discrepancies* between timed automata and ECA:

1. First, we show that *there is no finite time abstract language equivalence* on the valuations of *event clocks*, whereas the region equivalence [2] *is* a finite time abstract language equivalence for timed automata. This implies, in particular, that *regions do not form a finite time-abstract bisimulation for* ECA , thereby contradicting a claim found in the original paper on ECA [3].
2. With timed automata, checking language emptiness can be done by building the so-called region automaton [2] which recognizes $\mathsf{Untime}(L(A))$, the untimed version of A's timed language. A consequence of the surprising result of point 1 is that, for some ECA A, the *region automaton recognizes a strict subset of* $\mathsf{Untime}(L(A))$. Thus, the region automaton (as defined in [2]) *is not a sound construction for checking language emptiness of* ECA . We show however that a slight modification of the original definition (that we call the *existential region automaton*) allows to recover $\mathsf{Untime}(L(A))$. Unlike the timed automata case, our proof cannot rely on bisimulation arguments, and requires original techniques.
3. Efficient algorithms to analyze timed automata are best implemented using *zones* [1], that are in turn represented by *DBMs* [9]. Unfortunately, zones and DBMs cannot be directly applied to ECA. Indeed, a zone is, roughly speaking, a conjunction of constraints of the form $x - y \prec c$, where x, y are clocks, $\prec$ is either $<$ or $\leq$ and c is an integer. This makes sense in the case of timed automata, since the difference of two clock values is an invariant with time elapsing. This is not the case when we consider event clocks, as *prophecy and history clocks evolve in opposite directions with time elapsing*. Thus, we introduce the notions of event-zones and Event DBMs

that can handle constraints of the form $x + y \prec c$, when x and y are of different types.

4. In the case of timed automata two basic, zone-based algorithms for solving language emptiness have been studied: the *forward analysis algorithm* that iteratively computes all the states reachable from the initial state, and the *backward analysis algorithm* that computes all the states that can reach an accepting state. While the former might not terminate in general, the latter is guaranteed to terminate [1]. We show that this is not the case anymore with ECA: *both algorithms might not terminate* again because of event clocks evolving in opposite directions.

These observations reflect the structure of the paper. We close it by discussing the possibility to define *widening operators*, adapted from the *closure by region*, and the k-*approximation* that have been defined for timed automata [6]. The hardest part of this future work will be to obtain a proof of correctness for these operators, since, here again, we will not be able to rely on bisimulation arguments.

Remark Due to lack of space, most proofs have been omitted and can be found in a companion technical report.

2 Preliminaries

Words and timed words An alphabet Σ is a finite set of symbols. A (finite) *word* is a finite sequence $w = w_0 w_1 \cdots w_n$ of elements of Σ. We denote the length of w by $|w|$. We denote by Σ^* the set of words over Σ. A *timed word* over Σ is a pair $\theta = (\tau, w)$ such that w is a word over Σ and $\tau = \tau_0 \tau_1 \cdots \tau_{|w|-1}$ is a word over $\mathbb{R}^{\geq 0}$ with $\tau_i \leq \tau_{i+1}$ for all $0 \leq i < |w| - 1$. We denote by $\mathsf{T}\Sigma^*$ the set of timed words over Σ. A (timed) *language* is a set of (timed) words. For a timed word $\theta = (\tau, w)$, we let $\mathsf{Untime}(\theta) = w$. For a timed language L, we let $\mathsf{Untime}(L) = \{\mathsf{Untime}(\theta) \mid \theta \in L\}$.

Event clocks Given an alphabet Σ, we define the set of associated *event clocks* $\mathbb{C}_\Sigma = \mathbb{H}_\Sigma \cup \mathbb{P}_\Sigma$, where $\mathbb{H}_\Sigma = \{\overleftarrow{x_\sigma} \mid \sigma \in \Sigma\}$ is the set of *history clocks*, and $\mathbb{P}_\Sigma = \{\overrightarrow{x_\sigma} \mid \sigma \in \Sigma\}$ is the set of *prophecy clocks*. A *valuation* of a set of clocks is a function $v : C \to \mathbb{R}^{\geq 0} \cup \{\bot\}$, where $\bot$ means that the clock value is undefined. We denote by $\mathcal{V}(C)$ the set of all valuations of the clocks in C. For a valuation $v \in \mathcal{V}(C)$, for all $x \in \mathbb{H}_\Sigma$, we let $\langle v_1(x) \rangle = \lceil v(x) \rceil - v(x)$ and for all $x \in \mathbb{P}_\Sigma$, we let $\langle v(x) \rangle = v(x) - \lfloor v(x) \rfloor$, where $\lfloor v(x) \rfloor$ and $\lceil v(x) \rceil$ denote respectively the largest previous and smallest following integer. We also denote by $v^\pm$ the valuation s.t. $v^\pm(x) = v(x)$ for all $x \in \mathbb{H}_\Sigma$, and $v^\pm(x) = -v(x)$ for all $x \in \mathbb{P}_\Sigma$.

For all valuation $v \in \mathcal{V}(C)$ and all $d \in \mathbb{R}^{\geq 0}$ such that $v(x) \geq d$ for all $x \in \mathbb{P}_\Sigma \cap C$, we define the valuation $v + d$ obtained from v by letting d time units elapse: for all $x \in \mathbb{H}_\Sigma \cap C$, $(v + d)(x) = v(x) + d$ and for all $x \in \mathbb{P}_\Sigma \cap C$, $(v + d)(x) = v(x) - d$, with the convention that $\bot + d = \bot - d = \bot$. A valuation is *initial* iff $v(x) = \bot$ for all $x \in \mathbb{H}_\Sigma$, and *final* iff $v(x) = \bot$ for all $x \in \mathbb{P}_\Sigma$. We note $v[x := c]$ the valuation that matches v on all its clocks except for $v(x)$ that equals c.

An *atomic clock constraint* over $C \subseteq \mathbb{C}_\Sigma$ is either true or of the form $x \sim c$, where $x \in C$, $c \in \mathbb{N}$ and $\sim \in \{<, >, =\}$. A *clock constraint* over C is a Boolean combination of atomic clock constraints. We denote $\mathsf{Constr}(C)$ the set of all possible clock constraints over C. A valuation $v \in \mathcal{V}(C)$ satisfies a clock constraint $\psi \in \mathsf{Constr}(C)$, denoted $v \models \psi$ according to the following rules: $v \models \mathsf{true}$, $v \models x \sim c$ iff $v(x) \sim c$, $v \models \neg\psi$ iff $v \not\models \psi$, and $v \models \psi_1 \wedge \psi_2$ iff $v \models \psi_1$ and $v \models \psi_2$.

Event-clock automata An *event-clock automaton* $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$ (ECA for short) is a tuple, where Q is a finite set of locations, $q_i \in Q$ is the initial location, Σ is an alphabet, $\delta \subseteq Q \times \Sigma \times \mathsf{Constr}(\mathbb{C}_\Sigma) \times Q$ is a finite set of edges, and $\alpha \subseteq Q$ is the set of accepting locations. We additionally require that, for each $q \in Q$, $\sigma \in \Sigma$, δ is defined for a finite number of $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$. An *extended state* (or simply state) of an ECA $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$ is a pair (q, v) where $q \in Q$ is a location, and $v \in \mathcal{V}(\mathbb{C}_\Sigma)$ is a valuation.

Runs and accepted language The semantics of an ECA $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$ is best described by an infinite transition system $\mathsf{TS}_A = \langle Q^A, Q_i^A, \rightarrow, \alpha^A \rangle$, where $Q^A = Q \times \mathcal{V}(\mathbb{C}_\Sigma)$ is the set of extended states of A, $Q_i^A = \{(q_i, v) \mid v \text{ is initial}\}$, $\alpha^A = \{(q, v) \mid q \in \alpha \text{ and } v \text{ is final}\}$. The transition relation $\rightarrow \subseteq Q^A \times \mathbb{R}^{\geq 0} \times Q^A \cup Q^A \times \Sigma \times Q^A$ is s.t. (i) $\big((q, v), t, (q, v')\big) \in \rightarrow$ iff $v' = v + t$ (we denote this by $(q, v) \xrightarrow{t} (q, v')$), and (ii) $\big((q, v), \sigma, (q', v')\big) \in \rightarrow$ iff there is $(q, \sigma, \psi, q') \in \delta$ and $\overline{v} \in \mathcal{V}(\mathbb{C}_\Sigma)$ s.t. $\overline{v}[\overrightarrow{x_\sigma} := 0] = v$, $\overline{v}[\overleftarrow{x_\sigma} := 0] = v'$ and $\overline{v} \models \psi$ (we denote this $(q, v) \xrightarrow{\sigma} (q', v')$). We note $(q, v) \xrightarrow{t,\sigma} (q', v')$ whenever there is (q'', v'') s.t. $(q, v) \xrightarrow{t} (q'', v'') \xrightarrow{\sigma} (q', v')$. Intuitively, this means that an history clock $\overleftarrow{x_\sigma}$ always records the time elapsed since the last occurrence of the corresponding σ event, and that a prophecy clock $\overrightarrow{x_\sigma}$ always predicts the delay up to the next occurrence of σ. Thus, when firing a σ-labeled transition, the guard must be tested against $\overline{v}$ (as defined above) because it correctly predicts the next occurrence of σ and correctly records its last occurrence (unlike v and v', as $v(\overrightarrow{x_\sigma}) = 0$ and $v'(\overleftarrow{x_\sigma}) = 0$)

A sequence $(q_0, v_0)(t_0, w_0)(q_1, v_1)(t_1, w_1)(q_2, v_2) \cdots (q_n, v_n)$ is a (q, v)-*run* of A on the timed word $\theta = (\tau, w)$ iff: $(q_0, v_0) = (q, v)$, $t_0 = \tau_0$, for any $1 \leq i \leq n-1$: $t_i = \tau_i - \tau_{i-1}$, and for any $0 \leq i \leq n-1$: $(q_i, v_i) \xrightarrow{t_i, w_i} (q_{i+1}, v_{i+1})$. A (q, v)-run is *initialized* iff $(q, v) \in Q_i^A$ (in this case, we simply call it a run). A (q, v)-run on θ, ending in (q_n, v_n) is *accepting* iff $(q_n, v_n) \in \alpha^A$. In this case, we say that the run accepts θ. For an ECA A and an extended state (q, v) of A, we denote by $L(A, (q, v))$ the set of timed words accepted by a (q, v)-run of A, and by $L(A)$ the set of timed words accepted by an initialized run of A.

3 Equivalence relations for event-clocks

A classical technique to analyze timed transition systems is to define *time abstract equivalence relations* on the set of states, and to reason on the *quotient* transition system. In the case of *timed automata*, a fundamental concept is the *region equivalence* [2], which is a *finite time-abstract* bisimulation, and allows to decide properties of timed automata such as reachability. Contrary to a widespread belief [3], we show that the class

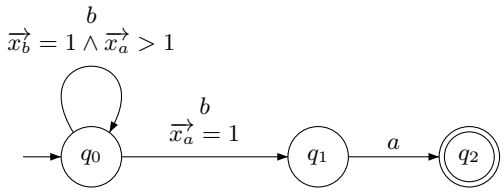

Fig. 1. The automaton A_{inf}

of ECA does not benefit of these properties, as ECA **admit no finite time-abstract language equivalence**.

Time-abstract equivalence relations Let $\mathcal{C}$ be a class of ECA, all sharing the same alphabet Σ. We recall three equivalence notions on event clock valuations:

- $\lesssim \subseteq \mathcal{V}(\mathbb{C}_\Sigma) \times \mathcal{V}(\mathbb{C}_\Sigma)$ is a *time abstract simulation relation* for the class $\mathcal{C}$ iff, for all $\mathcal{A} \in \mathcal{C}$, for all location q of $\mathcal{A}$, for all $(v_1, v_2) \in \lesssim$, for all $t_1 \in \mathbb{R}^{\geq 0}$, for all $a \in \Sigma$: $(q, v_1) \xrightarrow{t_1,a} (q', v_1')$ implies that there exists $t_2 \in \mathbb{R}^{\geq 0}$ s.t. $(q, v_2) \xrightarrow{t_2,a} (q', v_2')$ and $v_1' \lesssim v_2'$. In this case, we say that v_2 *simulates* v_1. Finally, $\simeq \subseteq \mathcal{V}(\mathbb{C}_\Sigma) \times \mathcal{V}(\mathbb{C}_\Sigma)$ is a *time abstract simulation equivalence* iff there exists a time abstract simulation relation $\lesssim$ s.t. $\simeq = \{(v_1, v_2) \mid v_1 \lesssim v_2 \text{ and } v_2 \lesssim v_1\}$
- $\sim$ is a *time abstract bisimulation equivalence* for the class $\mathcal{C}$ iff it is a *symmetric* time abstract simulation for the class $\mathcal{C}$.
- $\approx_L \subseteq \mathcal{V}(\mathbb{C}_\Sigma) \times \mathcal{V}(\mathbb{C}_\Sigma)$ is a *time abstract language equivalence* for the class $\mathcal{C}$ iff for all $\mathcal{A} \in \mathcal{C}$, for all location q of $\mathcal{A}$, for all $(v_1, v_2) \in \approx_L$: $\mathsf{Untime}(L(q, v_1)) = \mathsf{Untime}(L(q, v_2))$

We say that an equivalence relation is *finite* iff it is of finite index. Clearly, any time abstract bisimulation is a time abstract simulation equivalence, and any time abstract simulation equivalence is a time abstract language equivalence. We prove the absence of *finite* time abstract language equivalence for ECA, thanks to A_{inf} depicted in Fig. 1:

Proposition 1. *There is no finite time abstract language equivalence for* ECA.

Proof (Sketch). Assume $\approx_L$ is a time abstract language equivalence on event-clock valuations. For any $n \in \mathbb{N}$, let v^n denote the *initial* valuation of $\mathbb{C}_{\{a,b\}}$ s.t. $v^n(\overrightarrow{x_a}) = n$ and $v^n(\overrightarrow{x_b}) = 0$, and let θ^n be the timed word $(b,0)(b,1)(b,2)\cdots(b,n-1)(a,n)$. Consider the automaton A_{inf} in Fig. 1 and observe that for all $n \geq 0$, $\mathsf{Untime}(\mathsf{L}(A_{\mathsf{inf}}, (q_0, v^n))) = \mathsf{Untime}(\{\theta^n\}) = \{b^n a\}$. Hence for all the (infinitely many) pairs (i, j) with $i \neq j$: $v^i \not\approx_L v^j$, and thus $\approx_L$ is not finite. □

Corollary 2. *There is no* finite *time abstract language equivalence, no* finite *time abstract simulation equivalence and no* finite *time abstract bisimulation for* ECA.

4 Regions and event clocks

For the class of timed automata, the *region equivalence* has been shown to be a *finite time-abstract bisimulation*, which is used to build the so-called *region automaton*, a finite-state automaton recognizing $\mathsf{Untime}(L(A))$ for all timed automata A [2]. Corollary 2 tells us that regions are not a time-abstract bisimulation for ECA (contrary to what was claimed in [3]). Let us show that we can nevertheless rely on the notion of region to build a finite automaton recognizing $\mathsf{Untime}(L(A))$ for all ECA A.

Regions Let us fix a set of clocks $C \subseteq \mathbb{C}_\Sigma$ and a constant $cmax \in \mathbb{N}$. We first recall two region equivalences from the literature. The former, denoted $\approx_{cmax}$, is the classical Alur-Dill region equivalence for timed automata [2] while the latter (denoted $\approx^{\angle}_{cmax}$) is adapted from Bouyer [6] and refines the former:

- For any $v_1, v_2 \in \mathcal{V}(C)$: $v_1 \approx_{cmax} v_2$ iff:
  - (C1) for all $x \in C$, $v_1(x) = \bot$ iff $v_2(x) = \bot$,
  - (C2) for all $x \in C$: either $v_1(x) > cmax$ and $v_2(x) > cmax$, or $\lceil v_1(x) \rceil = \lceil v_2(x) \rceil$ and $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$,
  - (C3) for all $x_1, x_2 \in C$ s.t. $v_1(x_1) \leq cmax$ and $v_1(x_2) \leq cmax$: $\langle v_1(x_1) \rangle \leq \langle v_1(x_2) \rangle$ if and only if $\langle v_2(x_1) \rangle \leq \langle v_2(x_2) \rangle$.
- For all $v_1, v_2 \in \mathcal{V}(C)$: $v_1 \approx^{\angle}_{cmax} v_2$ iff: $v_1 \approx_{cmax} v_2$ and:
  - (C4) For all $x_1, x_2 \in C$ s.t. $v_1(x_1) > cmax$ or $v_1(x_2) > cmax$: either we have $\left|v_1^{\pm}(x_1) - v_1^{\pm}(x_2)\right| > 2 \cdot cmax$ and $\left|v_2^{\pm}(x_1) - v_2^{\pm}(x_2)\right| > 2 \cdot cmax$; or we have $\lfloor v_1^{\pm}(x_1) - v_1^{\pm}(x_2) \rfloor = \lfloor v_2^{\pm}(x_1) - v_2^{\pm}(x_2) \rfloor$ and $\lceil v_1^{\pm}(x_1) - v_1^{\pm}(x_2) \rceil = \lceil v_2^{\pm}(x_1) - v_2^{\pm}(x_2) \rceil$.

Equivalence classes of both $\approx_{cmax}$ and $\approx^{\angle}_{cmax}$ are called *regions*. We denote by $\mathsf{Reg}(C, cmax)$ and $\mathsf{Reg}^{\angle}(C, cmax)$ the set of regions of $\approx_{cmax}$ and $\approx^{\angle}_{cmax}$ respectively. Fig. 2 (a), (b) and (c) illustrate these two notions. Comparing (a) and (b) clearly shows how $\approx^{\angle}_{cmax}$ refines $\approx_{cmax}$ by introducing diagonal constraints between clocks larger than $cmax$. Moreover, (c) shows why we need to rely on $v_1^{\pm}$ and $v_2^{\pm}$ in C4: in this case, C contains an history and a prophecy clock that evolve in opposite directions with time elapsing. Thus, their *sum* remains constant over time (hence the $2 \cdot cmax$ in C4).

Observe that, for any $cmax$, and for any finite set of clocks C, $\mathsf{Reg}(C, cmax)$ and $\mathsf{Reg}^{\angle}(C, cmax)$ are *finite* sets. A region r on set of clocks C is *initial* (resp. *final*) iff it contains only initial (final) valuations.

Regions are not a language equivalence Since both notions of regions defined above are finite, Corollary 2 implies that they cannot form a language equivalence for ECA. Let us explain intuitively why it is not the case. Consider $\mathsf{Reg}\left(\mathbb{P}_{\{a,b\}}, 1\right)$ and the two valuations v_1 and v_2 in Fig. 2 (a). Clearly, v_1 can reach the region where $\overrightarrow{x_a} = 1$ and $\overrightarrow{x_b} > 1$, while v_2 cannot. Conversely, v_2 can reach $\overrightarrow{x_a} > 1$ and $\overrightarrow{x_b} = 1$ but v_2 cannot. It is easy to build an ECA with $cmax = 1$ that distinguishes between those two cases and accepts different words. Then, consider $\mathsf{Reg}^{\angle}\left(\mathbb{P}_{\{a,b\}}, 1\right)$ and the valuations v^3 and v^4 (not shown in the figure) s.t. $v^3(\overrightarrow{x_b}) = v^4(\overrightarrow{x_b}) = 1$, $v^3(\overrightarrow{x_a}) = 4$ and $v^4(\overrightarrow{x_a}) = 5$. It is easy to see that for A_{inf} in Fig. 1: $\mathsf{Untime}(L(A_{\mathrm{inf}}, (q_0, v^3))) = \{bbba\} \neq \{bbbba\} =$

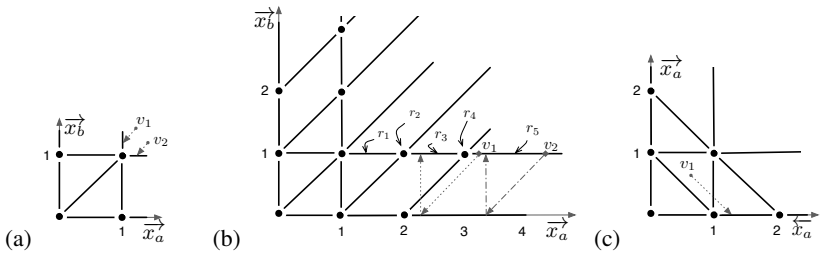

Fig. 2. The sets of regions (a) $\mathsf{Reg}\left(\mathbb{P}_{\{a,b\}},1\right)$, (b) $\mathsf{Reg}^{\angle}\left(\mathbb{P}_{\{a,b\}},1\right)$ and (c) $\mathsf{Reg}^{\angle}\left(\mathbb{C}_{\{a\}},1\right)$. Dotted arrows show the trajectories followed by the valuations with time elapsing. Curved arrows are used to refer to selected regions.

$\mathsf{Untime}(L(A_{\mathsf{inf}},(q_0,v^4)))$, although v^3 and v^4 belong to the same region. Indeed, from v^3, the (q_0,q_0) loop can be fired 3 times before we reach $\overrightarrow{x_a}=1$ and the (q_0,q_1) edge can be fired. However, the (q_0,q_0) loop has to be fired 4 times from v^4 before we reach $\overrightarrow{x_a}=1$ and the (q_0,q_1) edge can be fired. Remark that these are essentially the same arguments as in the proof of Proposition 1. These two examples illustrate the issue with *prophecy clocks* and regions. Roughly speaking, to keep the set of regions finite, valuations where the clocks are *too large* (for instance, $>cmax$ in the case of $\mathsf{Reg}\,(C,cmax)$) belong to the same region. This is not a problem for history clocks as an history clock larger than $cmax$ remains over $cmax$ with time elapsing. This is not the case for prophecy clocks whose values *decrease with time elapsing*: eventually, those clocks reach a value $\leq cmax$, but the region equivalence is too coarse to allow to predict the region they reach.

Region automata Let us now consider the consequence of Corollary 2 on the notion of region automaton. We first define two variants of the region automaton:

Definition 3. *Let $A=\langle Q,q_i,\Sigma,\delta,\alpha\rangle$ and $\mathcal{R}$ be a set of regions on $\mathcal{V}\left(\mathbb{C}_\Sigma\right)$. Then, the **existential** (resp. **universal**) $\mathcal{R}$-*region automaton of $\mathcal{A}$ *is the finite transition system $RA(\exists,\mathcal{R},A)$ (resp. $RA(\forall,\mathcal{R},A)$) defined by $\left\langle Q^R,Q_i^R,\Sigma,\delta^R,\alpha^R\right\rangle$ s.t.:*

1. $Q^R=Q\times\mathcal{R}$
2. $Q_i^R=\{(q_i,r)\mid r$ *is an initial region*$\}$
3. $\delta^R\subseteq Q^R\times\Sigma\times Q^R$ *is s.t.* $\left((q_1,r_1),a,(q_2,r_2)\right)\in\delta$ *iff* ***there exists a valuation*** *(resp.* ***for all valuations****)* $v_1\in r_1$*, there exists a time delay* $t\in\mathbb{R}^{\geq 0}$ *and a valuation* $v_2\in r_2$ *s.t.* $(q_1,v_1)\xrightarrow{t,a}(q_2,v_2)$.
4. $\alpha^R=\{(q,r)\mid q\in\alpha$ *and* r *is a final region*$\}$

Let $R=\left\langle Q^R,Q_i^R,\Sigma,\delta^R,\alpha^R\right\rangle$ be a region automaton and w be an (untimed) word over Σ. A *run* of R on $w=w_0w_1\dots w_n$ is a finite sequence $(q_0,r_0)(q_1,r_1)\dots(q_{n+1},r_{n+1})$ of states of R s.t.: $(q_0,r_0)\in Q_i^R$ and for all $0\leq i\leq n$: $\left((q_i,r_i),w_i,(q_{i+1},r_{i+1})\right)\in\delta^R$. Such a run is *accepting* iff $(q_{n+1},r_{n+1})\in\alpha^R$ (in that case, we say that w is accepted by R). The language $L(R)$ of R is the set of all untimed words accepted by R.

Let A be an ECA with alphabet Σ and maximal constant $cmax$. If we adapt and apply the notion of region automaton, as defined for timed automata [2], to A we obtain $RA(\forall, \mathsf{Reg}\left(\mathbb{C}_\Sigma, cmax\right), A)$. To alleviate notations, we denote it by $\mathsf{RegAut}_\forall\left(A\right)$. In the rest of the paper, we also consider three other variants: (i) $\mathsf{RegAut}_\forall^{\not<}\left(A\right) = RA(\forall, \mathsf{Reg}^{\not<}\left(\mathbb{C}_\Sigma, cmax\right), A)$, (ii) $\mathsf{RegAut}_\exists\left(A\right) = RA(\exists, \mathsf{Reg}\left(\mathbb{C}_\Sigma, cmax\right), A)$ and (iii) $\mathsf{RegAut}_\exists^{\not<}\left(A\right) = RA(\exists, \mathsf{Reg}^{\not<}\left(\mathbb{C}_\Sigma, cmax\right), A)$. Observe that, for timed automata, all these automata coincide, and thus accept the untimed language (this can be proved by a bisimulation argument) [2]. Let us see how these results adapt (or not) to ECA.

Recognized language of universal region automata Let us show that, in general *universal* region automata *do not recognize the untimed language of the* ECA.

Lemma 4. *There is an* ECA A *such that* $L(\mathsf{RegAut}_\forall\left(A\right)) \subsetneq \mathsf{Untime}(L(A))$ *and such that* $L(\mathsf{RegAut}_\forall^{\not<}\left(A\right)) \subsetneq \mathsf{Untime}(L(A))$.

Proof (Sketch). The ECA A_{inf} in Fig. 1 enjoys this property. We detail the arguments for the second case. Since $cmax = 1$, the set of regions we consider is depicted in Fig. 2 (b) (for the valuations where clocks are $\neq \bot$). Assume there is, in $\mathsf{RegAut}_\forall^{\not<}\left(A_{\mathsf{inf}}\right)$, an edge of the form $\big((q_0, r), b, (q_0, r')\big)$ were r is initial. This implies that $r' \in \{r_1, \ldots, r_5\}$ (we refer to the names in Fig. 2), because of the guard of the (q_0, q_0) loop. Since $\mathsf{Untime}(L(A_{\mathsf{inf}})) = \{b^n a \mid n \geq 1\}$, it must be possible to accept an arbitrary number of b's from one of the (q_0, r'). Let us show that it is not the case. From r_3 and r_4 we have edges $\big((q_0, r_3), b, (q_0, r_1)\big)$ and $\big(q_0, r_4), b, (q_0, r_2)\big)$. However, there is no valuation $v \in r_1 \cup r_2$ s.t. $(v + t)(\overrightarrow{x_b}) = 0$ and $(v + t)(\overrightarrow{x_a}) > 1$ for some t. Thus, there is, in $\mathsf{RegAut}_\forall^{\not<}\left(A_{\mathsf{inf}}\right)$, no edge of the form $\big((q_0, r), b, (q_0, r')\big)$ when $r \in \{r_1, r_2\}$. Finally, there is no edge of the form $\big((q_0, r_5), b, (q_0, r)\big)$ because *some valuations* of r_5 (such as v_1) will reach r_3 and *some others* (such as v_2) will stay in r_5 after the firing of the loop. Since we consider an *universal* automaton, (q_0, r_5) has no successor. □

Recognized language of existential region automata Fortunately, the definition of *existential region automaton* allows us to recover a finite transition system recognizing exactly $\mathsf{Untime}(L(A))$, for all ECA A. Remark that, to establish this result, we cannot rely on bisimulation arguments. Let us show that $\mathsf{Untime}(L(A)) \subseteq L(\mathsf{RegAut}_\exists^{\not<}\left(A\right)) \subseteq L(\mathsf{RegAut}_\exists\left(A\right)) \subseteq \mathsf{Untime}(L(A))$.

The two leftmost inequalities are easily established. Let $(q_0, v_0)(t_0, w_0)(q_1, v_1)$ $(t_1, w_1) \cdots (q_n, v_n)$ be an accepting run of A on $\theta = (\tau, w)$. Thus, $\theta \in L(A)$. For all $0 \leq i \leq n$ let r_i be the (unique) region containing v_i. Then, by definition of $\mathsf{RegAut}_\exists^{\not<}\left(A\right)$, $(q_0, r_0)w_0(q_1, r_1)w_1 \cdots (q_n, r_n)$ is an accepting run of $\mathsf{RegAut}_\exists^{\not<}\left(A\right)$ on $w = \mathsf{Untime}(\theta)$. Hence $\mathsf{Untime}(L(A)) \subseteq L(\mathsf{RegAut}_\exists^{\not<}\left(A\right))$. Second, since $\approx_{cmax}^{\not<}$ refines $\approx_{cmax}$, each accepting run $(q_0, r_0)w_0(q_1, r_1)w_1 \cdots (q_n, r_n)$ in $\mathsf{RegAut}_\exists^{\not<}\left(A\right)$ corresponds to an accepting run $(q_0, r'_0)w_0(q_1, r'_1)w_1 \cdots (q_n, r'_n)$ in $\mathsf{RegAut}_\exists\left(A\right)$, where for any $0 \leq i \leq n$, r'_i is the (unique) region of $\mathsf{Reg}\left(\mathbb{C}_\Sigma, cmax\right)$ that contains r_i. Hence, $L(\mathsf{RegAut}_\exists^{\not<}\left(A\right)) \subseteq L(\mathsf{RegAut}_\exists\left(A\right))$.

To establish $L(\mathsf{RegAut}_\exists\left(A\right)) \subseteq \mathsf{Untime}(L(A))$ we need to rely on the notion of *weak time successor*. The set of *weak time successors* of v by t time units is:

$$v +_{\mathrm{w}} t = \left\{ v' \middle| \forall x : \begin{array}{c} \big(x \in \mathbb{P}_\Sigma \text{ and } v(x) > cmax\big) \text{ implies } v'(x) > cmax - t \\ \text{and} \\ \big(x \notin \mathbb{P}_\Sigma \text{ or } v(x) \leq cmax \text{ or } v(x) = \bot\big) \text{ implies } v'(x) = (v + t)(x) \end{array} \right\}$$

As can be seen, weak time successors introduce non-determinism on prophecy clocks that are larger than $cmax$. So, $v +_{\mathrm{w}} t$ is a *set* of valuations. Let q be a location of an ECA. We write $(q, v) \xrightarrow{t}_w (q, v')$ whenever $v' \in (v +_{\mathrm{w}} t)$. Then, a sequence $(q_0, v_0)(t_0, w_0)$ $(q_1, v_1)(t_1, w_1)(q_2, v_2) \cdots (q_n, v_n)$ is an initialized *weak run*, on $\theta = (\tau, w)$, of an ECA $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$ iff $q_0 = q_i$, v_0 is initial, $t_0 = \tau_0$, for any $1 \leq i \leq n-1$: $t_i = \tau_i - \tau_{i-1}$, and for any $0 \leq i \leq n-1$: there is (q'_i, v'_i) s.t. $(q_i, v_i) \xrightarrow{t_i}_w (q'_i, v'_i) \xrightarrow{w_i} (q_{i+1}, v_{i+1})$. A weak run is accepting iff $q_n \in \alpha$ and v_n is final. The weak language $\mathsf{wL}(A)$ of A is the set of all timed words θ s.t. there is an accepting weak run on θ. Clearly, $L(A) \subseteq \mathsf{wL}(A)$ as every run is also a weak run. However, the converse also holds, since the non-determinism appears only on clocks larger than $cmax$, which the automaton cannot distinguish:

Proposition 5. *For any* ECA A*:* $L(A) = \mathsf{wL}(A)$.

Then, we prove that *weak time successors* enjoy a property which is reminiscent of time abstract bisimulation. This allows to establish Theorem 7.

Lemma 6. *Let* C *be a set of clocks and let* $cmax$ *be a natural constant. For any* $v_1, v_2 \in \mathcal{V}(C)$ *s.t.* $v_1 \approx_{cmax} v_2$*, for any* $t_1 \in \mathbb{R}^{\geq 0}$*, there exist* t_2 *and* $v' \in (v_2 +_{\mathrm{w}} t_2)$ *s.t.* $v_1 + t_1 \approx_{cmax} v'$.

Theorem 7. *For any* ECA $A = (\Sigma, Q, q_i, \delta, \alpha)$*:* $\mathsf{L}(\mathsf{RegAut}_{\exists}(A)) \subseteq \mathsf{Untime}(\mathsf{L}(A))$.

Proof (Sketch). For every run on w in the region automaton, we build a sequence of time stamps τ s.t. $\theta = (\tau, w)$ is in $L(A)$. The main difficulty stems from the fact that we consider an *existential automaton*: assume there are $\big((q_1, r_1), a, (q_2, r_2)\big)$ and $\big((q_2, r_2), b, (q_3, r_3)\big)$ in δ^R. Then, there are $v_1 \in r_1$, $v'_1, v_2 \in r_2$ and $v'_2 \in r_3$ s.t. $(q_1, v_1) \xrightarrow{a,t_1} (q_2, v'_1)$ and $(q_2, v_2) \xrightarrow{b,t_2} (q_3, v'_2)$ for some t_1, t_2, but *possibly with* $v'_1 \neq v_2$. Building θ by induction is thus more involved than with a universal automaton. However, for any run of $\mathsf{RegAut}_{\exists}(A)$ over a word $w \in \Sigma^*$, we can inductively build, by using Lemma 6, a time sequence τ and a *weak run* of A over (τ, w) that visits the same locations. By Proposition 5, this concludes the proof. □

Size of the existential region automaton The number of Alur-Dill regions on n clocks and with maximal constant $cmax$ is at most $R(n, cmax) = n! \times 2^n \times (2 \times cmax + 2)^n$ [2]. Adapting this result to take into account the $\perp$ value, we have: $|\mathsf{Reg}(\mathbb{C}_\Sigma, cmax)| \leq R(2 \times |\Sigma|, cmax + 1)$. Hence, the number of locations of $\mathsf{RegAut}_{\exists}(A)$ for an ECA A with m locations and alphabet Σ is at most $m \times R(2 \times |\Sigma|, cmax+1)$. In [3], a technique is given to obtain a finite automaton recognizing $\mathsf{Untime}(L(A))$ for all ECA A: first transform A into a non-deterministic timed automaton [2] A' s.t. $L(A') = L(A)$, then compute the region automaton of A'. However, building A' incurs a blow up in the number of clocks and locations, and the size of the region automaton of A' is at most $m \times 2^K \times R(K, cmax)$ where $K = 6 \times |\Sigma| \times (cmax + 2)$ is an upper bound on the number of atomic clock constraints in A. Our construction thus yields a smaller automaton.

5 Zones and event-clocks

In the setting of timed automata, the *zone datastructure* [9] has been introduced as an effective way to improve the running time and memory consumption of on-the-fly algorithms for checking emptiness. In this section, we *adapt* this notion to the framework of ECA, and discuss forward and backward analysis algorithms. Roughly speaking, a *zone* is a symbolic representation for a set of clock valuations that are defined by constraints of the form $x - y \prec c$, where x, y are clocks, $\prec$ is either $<$ or $\leq$, and c is an integer constant. Keeping the difference between clock values makes sense in the setting of timed automata as all the clocks have always real values and the difference between two clock values is an invariant over the elapsing of time. To adapt the notion of zone to ECA, we need to overcome two difficulties. First, prophecy and history clocks evolve in different directions with time elapsing. Hence, it is not always the case that if $v(x) - v(y) = c$ then $(v + t)(x) - (v + t)(y) = c$ for all t (for instance if x is a prophecy clocks and y an history clock). However, the *sum* of clocks of different types is now an invariant, so event clock zones must be definable, either by constraints of the form $x - y \prec c$, if x and y are both history or both prophecy clocks, or by constraints of the form $x + y \prec c$ otherwise. Second, clocks can now take the special value $\perp$. Formally, we introduce the notion of event-zone as follows.

Definition 8. *For a set C of clocks over an alphabet Σ, an* event-zone *is a subset of $\mathcal{V}(C)$ that is defined by a conjunction of constraints of the form $x = \perp$; $x \sim c$; $x_1 - x_2 \sim c$ if $x_1, x_2 \in \mathbb{H}_\Sigma$ or $x_1, x_2 \in \mathbb{P}_\Sigma$; and $x_1 + x_2 \sim c$ if either $x_1 \in \mathbb{H}_\Sigma$ and $x_2 \in \mathbb{P}_\Sigma$ or $x_1 \in \mathbb{P}_\Sigma$ and $x_2 \in \mathbb{H}_\Sigma$, with $x, x_1, x_2 \in C$, $\sim \in \{\leq, \geq, <, >\}$ and $c \in \mathbb{Z}$.*

Event-clock Difference Bound Matrices In the context of timed automata, Difference Bound Matrices (DBMs for short) have been introduced to represent and manipulate zones [5, 9]. Let us now adapt DBMs to event clocks.

Formally, an EDBM M of the set of clocks $C = \{x_1, \ldots, x_n\}$ is a $(n + 1)$ square matrix of elements from $\big(\mathbb{Z} \times \{<, \leq\}\big) \cup \{(\infty, <), (\perp, =), (?, =)\}$ s.t. for all $0 \leq i, j, \leq n$: $m_{i,j} = (\perp, =)$ implies $i = 0$ or $j = 0$ (i.e., $\perp$ can only appear in the first position of a row or column). Thus, a constraint of the form $x_i = \perp$ will be encoded with either $m_{i,0} = (\perp, =)$ or $m_{0,i} = (\perp, =)$. As in the case of DBMs, we assume that the extra clock x_0 is always equal to zero. Moreover, since prophecy clocks decrease with time evolving, they are encoded by their *opposite value* in the matrix. Hence the EDBM naturally encodes *sums* of variables when the two clocks are of different types. Each element $(m_{ij}, \prec_{ij})$ of the matrix thus represents either the constraint $x_i - x_j \prec_{ij} m_{ij}$ or the constraint $x_i + x_j \prec_{ij} m_{ij}$, depending on the type of x_i and x_j. Finally, the special symbol ? encodes the fact that the variable is not constrained (it can take any real value, or the $\perp$ value). Formally, an EDBM M on set of clocks $C = \{x_1, \ldots, x_n\}$ represents the zone $[\![M]\!]$ on set of clocks C s.t. $v \in [\![M]\!]$ iff for all $0 \leq i, j \leq n$: **if** $M_{i,j} = (c, \prec)$ with $c \neq ?$ **then** $v^{\pm}(x_i) - v^{\pm}(x_j) \prec c$ (assuming $v^{\pm}(x_0)$ denotes the value 0 and assuming that for all $k \in \mathbb{Z} \cup \{\perp\}$: $\perp + k = \perp - k = k + \perp = k - \perp = \perp$). When $[\![M]\!] = \varnothing$, we say that M is *empty*. In the sequel, we also rely on the $\leq$ ordering on EDBM elements. We let $(m; \prec) \leq (m'; \prec')$ iff one of the following holds: either (i) $m' = ?$; or (ii) $m, m' \in \mathbb{Z} \cup \{\infty\}$ and $m < m'$; or (iii) $m = m'$ and either $\prec = \prec'$ or $\prec' = \leq$.

As an example, consider the two following EDBMs that both represent $x_1 = \bot \wedge 0 < x_3 - x_4 < 1 \wedge x_2 + x_4 \leq 2$ (where x_1, x_2 are prophecy clocks, and x_3, x_4 are history clocks):

$$\begin{pmatrix} (0,\leq) & (\bot,=) & (?,=) & (?,=) & (?,=) \\ (\bot,=) & (?,=) & (?,=) & (?,=) & (?,=) \\ (0,\leq) & (?,=) & (0,\leq) & (?,=) & (?,=) \\ (?,=) & (?,=) & (?,=) & (0,\leq) & (1,<) \\ (?,=) & (?,=) & (2,\leq) & (0,<) & (0,\leq) \end{pmatrix} \quad \begin{pmatrix} (0,\leq) & (\bot,=) & (\infty,<) & (0,\leq) & (0,\leq) \\ (\bot,=) & (?,=) & (?,=) & (?,=) & (?,=) \\ (0,\leq) & (?,=) & (0,\leq) & (0,\leq) & (0,\leq) \\ (\infty,<) & (?,=) & (\infty,<) & (0,\leq) & (1,<) \\ (\infty,<) & (?,=) & (2,\leq) & (0,<) & (0,\leq) \end{pmatrix}$$

Normal form EDBMs As in the case of DBMs, we define a *normal form* for EDBM, and show how to turn any EDBM M into a normal form EDBM M' s.t. $[\![M]\!] = [\![M']\!]$. A non-empty EDBM M is in *normal form* iff the following holds: (i) for all $1 \leq i \leq n$: $M_{i,0} = (\bot,=)$ iff $M_{0,i} = (\bot,=)$ and $M_{i,0} = (?,=)$ iff $M_{0,i} = (?,=)$, (ii) for all $1 \leq i \leq n$: $M_{i,0} \in \{(\bot,=),(?,=)\}$ implies $M_{i,j} = M_{j,i} = (?,=)$ for all $1 \leq j \leq n$, (iii) for all $1 \leq i,j \leq n$: $M_{i,j} = (?,=)$ iff either $M_{i,0} \in \{(?,=),(\bot,=)\}$ or $M_{j,0} \in \{(?,=),(\bot,=)\}$ and (iv) the matrix M' is a *normal form DBM* [9], where M' is obtained by projecting away all lines $1 \leq i \leq n$ s.t. $M_{i,0} \in \{(?,=),(\bot,=)\}$ and all columns $1 \leq j \leq n$ s.t. $M_{0,j} \in \{(?,=),(\bot,=)\}$ from M. To canonically represent the empty zone, we select a particular EDBM $M_\varnothing$ s.t. $[\![M_\varnothing]\!] = \varnothing$. For example, the latter EDBM of the above example is in normal form.

Then, given an EDBM M, Algorithm 1 allows to compute a normal form EDBM M' s.t. $[\![M]\!] = [\![M']\!]$. This algorithm relies on the function `DBMNormalise(`M`,`S`)`, where M is an $(\ell+1)\times(\ell+1)$ EDBM, and $S \subseteq \{0,\ldots,\ell\}$. `DBMNormalise(`$M$`,`$S$`)` applies the classical normalisation algorithm for DBMs [9] on the DBM obtained by projecting away from M all the lines and columns $i \notin S$. Algorithm 1 proceeds in three steps. In the first loop, we look for lines (resp. columns) i s.t. $M_{i,0}$ (resp. $M_{0,i}$) is $(\bot,=)$, meaning that there is a constraint imposing that $x_i = \bot$. In this case, the corresponding $M_{0,i}$ (resp. $M_{i,0}$) must be equal to $(\bot,=)$ too, and all the other elements in the ith line and column must contain $(?,=)$. If we find a j s.t. $M_{i,j} \neq (?,=)$ or $M_{j,i} \neq (?,=)$, then the zone is empty, and we return $M_\varnothing$. Then, in the second loop, the algorithm looks for lines (resp. columns) i with the first element equal to $(?,=)$ but containing a constraint of the form $(c,\prec)$, which imposes that the variable i must be different from $\bot$. We record this information by replacing the $(?,=)$ in $M_{i,0}$ (resp. $M_{0,i}$) by the weakest possible constraint that forces x_i to have a value different from $\bot$. This is either $(0,\leq)$ or $(\infty,<)$, depending on the type of x_i and is taken care by the `SetCst()` function. At this point the set S contains the indices of all variables that are constrained to be real. The algorithm finishes by calling the normalisation function for DBMs. Remark, in particular, that the algorithm returns $M_\varnothing$ iff M is empty which also provides us with a test for EDBM emptiness.

Proposition 9. *For all EDBM* M, `EDBMNormalise(`M`)` *returns a normal form EDBM* M' *s.t.* $[\![M']\!] = [\![M]\!]$.

Operations on zones The four basic operations we need to perform on event-zones are: (i) *future* of an event-zone Z : $\overrightarrow{Z} = \{v \in \mathcal{V}(\mathbb{C}_\Sigma) \mid \exists v' \in Z, t \in \mathbb{R}^{\geq 0} : v =$

```
1  EDBMNormalise(M) begin
2      Let S = {0} ;
3      foreach 1 ≤ i ≤ n s.t. M_{i,0} = (⊥,=) or M_{0,i} = (⊥,=) do
4          if ∃1 ≤ j ≤ n s.t. M_{i,j} ≠ (?,=) or M_{j,i} ≠ (?,=) then return M_∅;
5          M_{i,0} ← (⊥,=) ; M_{0,i} ← (⊥,=) ;
6      foreach 0 ≤ i,j ≤ n s.t. M_{i,j} ∉ {(?,=),(⊥,=)} do
7          S ← S ∪ {i,j} ;
8      foreach i,j ∈ S do SetCst(M_{i,j}) ;
9      M' ← DBMNormalise(M,S) ;
10     if M' = Empty then return M_∅ ;
11     return M' ;

12 SetCst(M_{i,j}) begin
13     if M_{i,j} = (?,=) then
14         if x_i ∈ P_Σ and (x_j ∈ H_Σ or x_j = x_0) then M_{i,j} ← (0,≤) ;
15         else M_{i,j} ← (∞,<) ;
```

Algorithm 1: A normalisation algorithm for EDBMs.

$v' + t\}$; (ii) *past* of an event-zone Z : $\overleftarrow{Z} = \{v \in \mathcal{V}(\mathbb{C}_\Sigma) \mid \exists t \in \mathbb{R}^{\geq 0} : v + t \in Z\}$; (iii) *intersection* of two event-zones Z and Z'; and (iv) *release* of a clock x in Z: $\mathsf{rel}_x(Z) = \{v[x := d] \mid v \in Z, d \in \mathbb{R}^{\geq 0} \cup \{\bot\}\}$. Moreover, we also need to be able to test for inclusion of two zones encoded as EDBMs. Let M, M_1 and M_2 be EDBMs in normal form, on n clocks. Then:

Future If $M = M_\varnothing$, we let $\overrightarrow{M} = M_\varnothing$. Otherwise, we let $\overrightarrow{M}$ be s.t.:

$$\overrightarrow{M}_{i,j} = \begin{cases} (0,\leq) & \text{if } M_{ij} \notin \{(\bot,=),(?,=)\},\ j = 0 \text{ and } x_i \in \mathbb{P}_\Sigma \\ (\infty,<) & \text{if } M_{ij} \notin \{(\bot,=),(?,=)\},\ j = 0 \text{ and } x_i \in \mathbb{H}_\Sigma \\ M_{i,j} & \text{otherwise} \end{cases}$$

Past If $M = M_\varnothing$, we let $\overleftarrow{M} = M_\varnothing$. Otherwise, we let $\overleftarrow{M}$ be s.t. for all i, j:

$$\overleftarrow{M}_{i,j} = \begin{cases} (\infty,<) & \text{if } M_{ij} \notin \{(\bot,=),(?,=)\},\ i = 0 \text{ and } x_j \in \mathbb{P}_\Sigma \\ (0,\leq) & \text{if } M_{ij} \notin \{(\bot,=),(?,=)\},\ i = 0 \text{ and } x_j \in \mathbb{H}_\Sigma \\ M_{i,j} & \text{otherwise} \end{cases}$$

Intersection We consider several cases. If $M^1 = M_\varnothing$ or $M^2 = M_\varnothing$, we let $M^1 \cap M^2 = M_\varnothing$. If there are $0 \leq i, j \leq n$ s.t. $M^1_{i,j} \not\leq M^2_{i,j}$ and $M^2_{i,j} \not\leq M^1_{i,j}$, we let $M^1 \cap M^2 = M_\varnothing$ too. Otherwise, we let $M^1 \cap M^2$ be the EDBM M' s.t for all i, j: $M'_{i,j} = min(M^1_{i,j}, M^2_{i,j})$.

Release Let x be an event clock. In the case where $M = M_\varnothing$, we let $\mathsf{rel}_x(M) = M_\varnothing$. Otherwise, we let $\mathsf{rel}_x(M)$ be the EDBM s.t. for all i, j:

$$\mathsf{rel}_x(M)_{i,j} = \begin{cases} M_{i,j} & \text{if } x_i \neq x \text{ and } x_j \neq x \\ (?,=) & \text{otherwise} \end{cases}$$

Inclusion We note $M^1 \subseteq M^2$ iff $M^1_{i,j} \leq M^2_{i,j}$ for all $0 \leq i, j \leq n$.

Proposition 10. *Let M, M^1, M^2 be EDBMs in normal form, on set of clocks C. Then, (i) $\overrightarrow{[\![M]\!]} = \left[\!\left[\overrightarrow{M}\right]\!\right]$, (ii) $\overleftarrow{[\![M]\!]} = \left[\!\left[\overleftarrow{M}\right]\!\right]$, (iii) $[\![M^1 \cap M^2]\!] = [\![M^1]\!] \cap [\![M^2]\!]$, (iv) for all clock $x \in C$, $\mathsf{rel}_x([\![M]\!]) = [\![\mathsf{rel}_x(M)]\!]$ and (v) $[\![M^1]\!] \subseteq [\![M^2]\!]$ iff $M^1 \subseteq M^2$.*

Forward and backward analysis We present now the forward and backward analysis algorithms adapted to ECA. From now on, we will consider an ECA $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$. We also let $\mathsf{Post}\left((q,v)\right) = \{(q',v') \mid \exists t, a : (q,v) \xrightarrow{t,a} (q',v')\}$ and $\mathsf{Pre}\left((q,v)\right) = \{(q',v') \mid \exists t, a : (q',v') \xrightarrow{t,a} (q,v)\}$ and we extend those operators to sets of states in the natural way. Moreover, given a set of valuations Z and a location q, we abuse notations and denote by (q,Z) the set $\{(q,v) \mid v \in Z\}$. Also, we let $\mathsf{Post}^*\left((q,Z)\right) = \bigcup_{n\in\mathbb{N}} \mathsf{Post}^n\left((q,Z)\right)$ and $\mathsf{Pre}^*\left((q,Z)\right) = \bigcup_{n\in\mathbb{N}} \mathsf{Pre}^n\left((q,Z)\right)$, where $\mathsf{Post}^0\left((q,Z)\right) = (q,Z)$ and $\mathsf{Post}^n\left((q,Z)\right) = \mathsf{Post}\left(\mathsf{Post}^{n-1}\left((q,Z)\right)\right)$, and similarly for $\mathsf{Pre}^n\left((q,Z)\right)$. The Post and Pre operators are sufficient to solve language emptiness for ECA:

Lemma 11 (adapted from [3], Lemma 1). *Let $A = \langle Q, q_i, \Sigma, \delta, \alpha \rangle$ be an* ECA*, let $I = \{(q_i, v) \mid v$ is initial$\}$, and let $\overline{\alpha} = \{(q,v) \mid q \in \alpha$ and v is final$\}$. Then:*
$$\mathsf{Post}^*\left(I\right) \cap \overline{\alpha} \neq \varnothing \text{ iff } \mathsf{Pre}^*\left(\overline{\alpha}\right) \cap I \neq \varnothing \text{ iff } L(A) \neq \varnothing.$$

Let us show how to compute these operators on event-zones. Given a location q, an event-zone Z on $\mathbb{C}_\Sigma$, and an edge $e = (q, a, \psi, q') \in \delta$, we let:

$$\mathsf{Post}_e\left((q_1, Z)\right) = \begin{cases} \left(q', \left(\mathsf{rel}_{\overleftarrow{x_a}}\left(\mathsf{rel}_{\overrightarrow{x_a}}(\overrightarrow{Z} \cap (\overrightarrow{x_a} = 0)) \cap \psi\right)\right) \cap (\overleftarrow{x_a} = 0)\right) & \text{if } q_1 = q \\ \varnothing & \text{otherwise} \end{cases}$$

$$\mathsf{Pre}_e\left((q_1, Z)\right) = \begin{cases} \left(q, \overleftarrow{\left(\mathsf{rel}_{\overrightarrow{x_a}}(\mathsf{rel}_{\overleftarrow{x_a}}(Z \cap (\overleftarrow{x_a} = 0)) \cap \psi)\right) \cap (\overrightarrow{x_a} = 0)}\right) & \text{if } q_1 = q' \\ \varnothing & \text{otherwise} \end{cases}$$

Then, it is easy to check that $\mathsf{Post}\left((q,Z)\right) = \cup_{e\in\delta}\mathsf{Post}_e\left((q,Z)\right)$ and that $\mathsf{Pre}\left((q,Z)\right) = \cup_{e\in\delta}\mathsf{Pre}_e\left((q,Z)\right)$. With the algorithms on EDBMs presented above, these definitions can be used to compute the Pre and Post of zones using their EDBM encodings. Remark that Pre and Post return *sets of event-zones* as these are not closed under union.

Let us now consider the ForwExact and BackExact algorithms to test for language emptiness of ECA, shown in Algorithm 2. In these two algorithms Z_0 denotes the zone $\bigwedge_{x\in\mathbb{H}_\Sigma} x = \bot$ containing all the possible initial valuations and Z_f denotes the zone $\bigwedge_{x\in\mathbb{P}_\Sigma} x = \bot$ representing all the possible final valuations. By Lemma 11, it is clear that ForwExact and BackExact are correct when they terminate. Unfortunately, Fig. 3 (a) shows an ECA on which the backward algorithm does not terminate. Since history and prophecy clocks are symmetrical, this example can be adapted to define an ECA on which the forward algorithm does not terminate either. Remark that in the case of timed automata, the forward analysis is not guaranteed to terminate, whereas the backward analysis *always terminates* (the proof relies on a bisimulation argument) [1].

Proposition 12. *Neither* ForwExact *nor* BackExact *terminate in general.*

```
1  ForwExact begin
2      Let Visited = ∅ ; Let Wait = {(q_i, Z_0)} ;
3      while Wait ≠ ∅ do
4          Get and remove (q, Z) from Wait ;
5          if q ∈ α and Z ⊆ Z_f then return Yes ;
6          if there is no (q, Z') ∈ Visited s.t. Z ⊆ Z' then
7              Visited := Visited ∪ {(q, Z)} ; Wait := Wait ∪ Post((q, Z)) ;
8      return No ;

9  BackExact begin
10     Let Visited = ∅ ; Let Wait = {(q, Z_f) | q ∈ α} ;
11     while Wait ≠ ∅ do
12         Get and remove (q, Z) from Wait ;
13         if q = q_i and Z ⊆ Z_0 then return Yes ;
14         if there is no (q, Z') ∈ Visited s.t. Z ⊆ Z' then
15             Visited := Visited ∪ {(q, Z)} ; Wait := Wait ∪ Pre((q, Z)) ;
16     return No ;
```

Algorithm 2: The forward and backward algorithms

Proof. We give the proof for BackExact, a similar proof for ForwExact can then be deduced by symmetry. Consider the ECA in Fig. 3 (a). Running the backward analysis algorithm from (q_2, Z_f), we obtain, after selecting the transition $e = (q_2, b, \texttt{true}, q_2)$, the zone $Z_1 = \overrightarrow{x_a} = \bot \wedge \overleftarrow{x_b} = 0$. Then, the transition $e' = (q_1, a, \overrightarrow{x_b} = 1, q_2)$ is back-firable and we attain the zone $Z_2 = \overrightarrow{x_a} \geq 0 \wedge \overrightarrow{x_b} \geq 1 \wedge \overrightarrow{x_b} - \overrightarrow{x_a} = 1$. At this point the transition $e'' = (q_1, a, \overleftarrow{x_a} = 1, q_1)$ is back-firable, which leads to the zone $Z_3 = \overrightarrow{x_b} \geq 1 \wedge \overrightarrow{x_a} \geq 0 \wedge 0 \leq \overleftarrow{x_a} \leq 1 \wedge \overrightarrow{x_b} - \overrightarrow{x_a} \geq 1 \wedge \overrightarrow{x_b} + \overleftarrow{x_a} \geq 2$. The back-firing of the e'' transition can be repeated, and, by induction, after n iterations of the loop, the algorithm reaches the zone $Z^n = \overrightarrow{x_b} \geq n \wedge \overrightarrow{x_a} \geq 0 \wedge 0 \leq \overleftarrow{x_a} \leq 1 \wedge \overrightarrow{x_b} - \overrightarrow{x_a} \geq n \wedge \overrightarrow{x_b} + \overleftarrow{x_a} \geq n + 1$. Thus, the condition of the **if** in line 14 is always fulfilled, and the algorithm visits an infinite number of zones, without reaching q_0. □

6 Future work: widening operators

As said earlier, the zone-based *forward* analysis algorithm does not terminate either in the case of timed automata. To recover termination, *widening operators* have been defined.The most popular widening operator is the so-called k-approximation on zones [**?**]. Roughly speaking, it is defined as follows: in the definition of the zone, replace any constraint of the form $x_i \prec c$ or $x_i - x_j \prec c$, by respectively $x_i < \infty$ and $x_i - x_j < \infty$ if and only if $c > k$, and replace any constraint of the form $c \prec x_i$ or $c \prec x_i - x_j$, by respectively $k < x_i$ and $k < x_i - x_j$, if and only if $c > k$. Such an operator can be easily computed on DBMs, and is a standard operation implemented in several tools such as as UppAal [4] for more more than 15 years. Nevertheless, this operator has been widely discussed in the recent literature since Bouyer has pointed out several flaws in

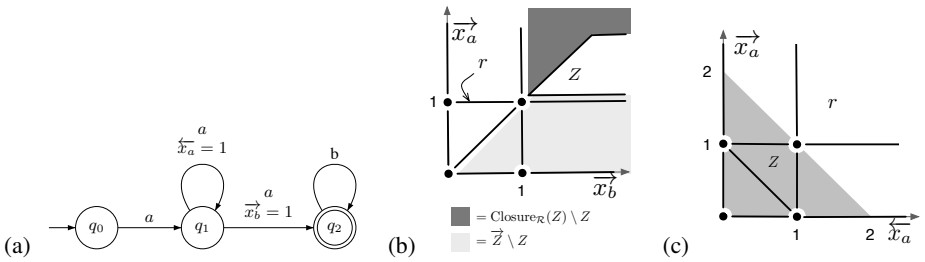

Fig. 3. (a): An ECA for which backward analysis does not terminate. (b) and (c): Examples for $\mathrm{Closure}_R$ and Approx_k

the proposed proofs of soundness [6]. Actually, the k-approximation is *sound* when the timed automaton contains *no diagonal constraints*. Unfortunately, k-approximation is *not sound* when the timed automaton contains diagonal constraints, and *no sound widening operator exists* in this case.

In [6], Bouyer identifies some subclasses of timed automata for which the widening operator is provably correct. The idea of the proof relies mainly on the definition of another widening operator, called the *closure by regions*, which is shown to be *sound*. The closure by regions of a zone Z, with respect to a set of regions $\mathcal{R}$ is defined as the smallest set of regions from $\mathcal{R}$ that have a non-empty intersection with Z, i.e. $\mathrm{Closure}_{\mathcal{R}}(Z) = \{r \in \mathcal{R} \mid Z \cap r \neq \varnothing\}$. Then, the proof concludes by showing that $\mathrm{Approx}_k(Z)$ is sound for some values of k (that are proved to exist) s.t.

$$Z \subseteq \mathrm{Approx}_k(Z) \subseteq \mathrm{Closure}_{\mathcal{R}}(Z). \tag{1}$$

In the perspective of bringing ECA from theory to implementation, *provably correct* widening operators are necessary, since neither the forward nor the backward algorithm terminate in general. We plan to adapt the k-approximation to ECA, and we believe that we can follow the general idea of the proof in [6]. However, the proof techniques will not be applicable in a straightforward way, for several reasons. First, the proof of [6] relies on the following property, which holds in the case of timed automata: for all zone Z and all location q: $\mathsf{Post}\left((q, \mathrm{Closure}_{\mathcal{R}}(Z))\right) \subseteq \mathrm{Closure}_{\mathcal{R}}(\mathsf{Post}\left((q, Z)\right))$. Unfortunately this is not the case in general with ECA. Indeed, consider the zone Z and the region r in Fig. 3 (b). Clearly, r is included in $\overrightarrow{\mathrm{Closure}_{\mathcal{R}}(Z)}$ but r is not included in $\mathrm{Closure}_{\mathcal{R}}(\overrightarrow{Z})$ (recall that prophecy clocks decrease with time elapsing). Moreover, the definition of the k approximation will need to be adapted to the case of ECA. Indeed, the second inclusion in (1) does not hold when using the k-approximation defined for timed automata, which merely replaces all constants $> k$ by ∞ in the constraints of the zone. Indeed, consider the event-zone Z defined by $\overleftarrow{x_a} + \overrightarrow{x_a} \leq 2$ in Fig. 3 (c), together with the set of regions $\mathcal{R} = \mathsf{Reg}\left(\mathbb{C}_{\{a\}}, 1\right)$. Clearly, with such a definition, the constraint $\overleftarrow{x_a} + \overrightarrow{x_a} \leq 2$ would be replaced by $\overleftarrow{x_a} + \overrightarrow{x_a} < \infty$, which yields an approximation that intersects with r, and is thus not contained in $\mathrm{Closure}_{\mathcal{R}}(Z)$. We keep open for future works the definition of a provably correct adaptation of the k-approximation for ECA.

References

1. R. Alur. Timed automata. In *Proceedings of CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer, 1999.
2. R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–236, 1994.
3. R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211(1-2):253–273, 1999.
4. G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. In *Proceedings of QEST'06*, pages 125–126. IEEE Computer Society, 2006.
5. R. Bellman. *Dynamic Programming*. Princeton university press, 1957.
6. P. Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, May 2004.
7. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Proceedings of CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998.
8. B. Di Giampaolo, G. Geeraerts, J. Raskin, and N. Sznajder. Safraless procedures for timed specifications. In *Proceedings of FORMATS'10*, volume 6246 of *Lecture Notes in Computer Science*, pages 2–22. Springer, 2010.
9. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.
10. C. Dima. Kleene theorems for event-clock automata. In *Proceedings of FCT'99*, volume 1684 of *Lecture Notes in Computer Science*, pages 215–225. Springer, 1999.
11. D. D'Souza and N. Tabareau. On timed automata with input-determined guards. In *Proccedings of FORMATS/FTRTFT'04*, volume 3253 of *Lecture Notes in Computer Science*, pages 68–83, 2004.
12. J.-F. Raskin and P.-Y. Schobbens. The logic of event clocks: decidability, complexity and expressiveness. *Automatica*, 34(3):247–282, 1998.
13. M. Sorea. Tempo: A model-checker for event-recording automata. In *Proceedings of RT-TOOLS'01*, Aalborg, Denmark, August 2001.
14. N. Tang and M. Ogawa. Event-clock visibly pushdown automata. In *Proceedings of SOFSEM'09*, volume 5404 of *Lecture Notes in Computer Science*, pages 558–569. Springer, 2009.