Distributed Event Clock Automata
Extended Abstract

James Ortiz[1] and Axel Legay[2,3] and Pierre-Yves Schobbens[1]

[1] Computer Science Faculty, University of Namur
[2] INRIA/IRISA, Rennes
[3] Institut Montefiore, University of Liège
james.ortizvega@fundp.ac.be
alegay@irisa.fr
pierre-yves.schobbens@fundp.ac.be

Abstract. In distributed real-time systems, we cannot assume that clocks are perfectly synchronized. To model them, we use independent clocks and define their timed semantics. The universal timed language, and the timed language inclusion of icTA are undecidable. Thus, we propose Recursive Distributed Event Clock Automata (DECA). DECA are closed under all boolean operations and their timed language inclusion problem is decidable (more precisely PSPACE-complete), allowing stepwise refinement. We also propose Distributed Event Clock Temporal Logic (DECTL), a real-time logic with independent time evolutions. This logic can be model-checked by translating a DECTL formula into a DECA automaton.

1 Introduction

Real-Time Distributed Systems (RTDS) take an increasingly important role in our society, including in aircrafts and spacecrafts, satellite telecommunication networks or positioning systems. Distributed Systems consist of computer systems at different locations, that communicate through a network to achieve their function. Real-Time Systems have to obey strict requirements about the time of their actions. To ensure these, they rely on clocks. When systems are widely distributed, we cannot assume that their clocks are perfectly synchronized.

One of the most successful techniques for modeling real-time systems are Timed Automata (TA) [2]. A timed automaton is a finite automaton augmented with real-valued clocks. Constraints on these clocks are used to restrict the behaviors of the automaton. The model of TA assumes perfect clocks: all clocks have infinite precision and are perfectly synchronized.

This causes TA to have an undecidable language inclusion problem [2]. The situation contrasts strongly with the one of automata without real time, where the problems of complementation, language inclusion, emptiness, union and intersection are decidable, as well as the satisfiability and validity of propositional linear temporal logic (LTL). These properties are the basis of the success of model-checking. When all these problems are decidable, we call the formalism

B. Bouchou-Markhoff et al. (Eds.): CIAA 2011, LNCS 6807, pp. 250–263, 2011.

(automata or logic) fully decidable. These negative results spurred a quest for expressive but still fully decidable formalisms. To restore decidability, [4] proposed to restrict the behavior of clocks. The key idea is that the problematic clocks of TA are reset by non-deterministic transitions. In contrast, an event clock (EC) x_p is reset when a given atomic proposition p occurs. The event clock values are deterministic and thus Event Clock Automata ECA are determinizable, making language inclusion decidable and thus enabling refinement based development. Event clocks can also be introduced in temporal logic [20]. An event clock constraint is naturally translated into a proposition $\lhd_I p$, that means "the last time that a p occurred was d time units ago, where d lies in I".

However, the expressiveness of ECA is rather weak. Furthermore, this logic violates the substitution principle: Any proposition should be replaceable by a formula. Therefore [12] introduced the notion of "recursive" event. In a recursive event model, the reset of a clock is decided by a lower-level automaton (or formula). This automaton cannot read the clock that it is resetting. Clock resets are thus still deterministic, but the concept of "event" is now much more expressive. $\rhd_I$ and $\lhd_I$ are modalities that can contain any subformulas, and can be nested. The temporal logic of recursive event clocks (variously called SCL [20] or EventClockTL [12]) has the same expressiveness as Metric Interval Temporal Logic MITL [3] (a decidable fragment of Metric Temporal Logic MTL where punctual constraints $\mathcal{U}_{\{k\}}$ are forbidden) in the interval semantics. First-and second-order monadic logics with matching expressiveness have been provided [12], yielding a natural, robust, fully decidable level of real-time expressiveness.

In this paper, we remove the assumption of perfect clock synchronization. Here, inspired by [6,14,1,10], we study the worst case: the clocks can advance totally independently if they are in different processes. [18,8] studied the opposite case, where the difference between clocks (drift) is infinitesimally small.

While [1] only studied untimed languages of their timed automata, namely the universal and existential languages, our first contribution is to define and study the corresponding timed languages (Section 4).

Our second contribution is to extend the Recursive Event Clock Automata (RECA) with distributed (a.k.a independent) clocks, yielding the Distributed Recursive Event Clock Automata (DECA). We will show that DECA are determinizable, thus closed under complementation, and thus that their language inclusion problem is decidable (more exactly, PSPACE-complete). We also show the decidability and regularity of their existential and universal timed languages (Section 5).

Our third contribution is to define a temporal logic with multiple observers, each with its own time evolution. This gives us the (Recursive) Distributed Event Clock Temporal Logic (DECTL), which is also PSPACE-complete (Section 6).

Structure of the paper. The rest of the paper is organized as follows. Sections 2 and 3 recall preliminary notions. Section 4 extends the semantics to timed languages. Section 5 defines DECA and studies their properties. Section 6 examines real-time temporal logics: it recalls EventClockTL [20], then introduces and studies DECTL. Due to space constraints, we only sketch proofs.

2 Preliminaries

We briefly recall the various models of time that are used in the literature [5]. We present our results in the interval semantics, that is the richest and most natural (but also most difficult) model. We also recall clocks and their constraints.

2.1 Models of Time

Models of time can be linear, considering a single future, or branching, considering several alternative futures. We only consider linear time in this paper. Our goal here is to model real-time systems, and thus we use the real numbers as our model of time. This avoids a premature commitment to a specific discretization of time. In this paper, we use the *interval semantics*, where the state of the model is known at any point in time, as opposed to *point semantics*, where it is known only at transitions.

Let $\mathbb{P}$ be a finite set of *(propositional) atoms*. A *letter* is an element of a finite set Σ. In this paper, we choose to define a letter as a propositional valuation over $\mathbb{P}$, so we pose $\Sigma = 2^{\mathbb{P}}$. Let $\mathbb{N}$ be the set of natural numbers, $\mathbb{R}$ denote the set of real numbers, $\mathbb{R}_{\geq 0}$ the set of non-negative real numbers. We denote by $\mathcal{I}(\mathbb{R}_{\geq 0})$ the set of real intervals whose bounds are in $\mathbb{R}_{\geq 0}$. An interval $I \in \mathcal{I}(\mathbb{R}_{\geq 0})$ is a convex subset of $\mathbb{R}_{\geq 0}$. An interval I is *contiguous* to I' when they are ordered: $I < I'$, and $I \cup I'$ is convex. An (alternating) interval sequence (AIS) is a monotone sequence $I = I_0 I_1 \cdots$ of non-empty intervals of $\mathcal{I}(\mathbb{R}_{\geq 0})$ where : (i) singular and open intervals alternate; (ii) $I_0 = \{0\}$; (iii) I_j is contiguous to I_{j+1}; (iv) if infinite, the sequence of intervals is *progressive*, i .e., for every $t \in \mathbb{R}_{\geq 0}$, there exists $j \in \mathbb{N}$ such that $t \in I_j$. An interval state sequence (ISS) on Σ is a pair $\theta = (\sigma, I)$ where $\sigma = \sigma_0 \sigma_1 \cdots$ is a (possibly infinite) word $\sigma \in \Sigma^{\leq \omega}$, and $I = I_0 I_1 \cdots$ is an AIS of the same length. This is the analog of a *timed word* [2]. An ISS can equivalently be seen as a sequence of pairs in $\Sigma \times \mathcal{I}(\mathbb{R}_{\geq 0})$. It can also be seen as a *signal*, i.e. a function from $\mathbb{R}_{\geq 0} \to \Sigma$: given $t \in \mathbb{R}_{\geq 0}$, let $i \in \mathbb{N}$ be the interval such that $t \in I_i$: We define $\theta(t)$ as σ_i. A signal derived from an ISS will always have finite variability. Below, our automata will consider two ISS θ_1, θ_2 that define the same signal as equivalent, noted $\theta_1 \equiv \theta_2$, even if the intervals might be split differently.

2.2 Clocks

A clock is a variable that increases with time. Thus, the value of a clock is the time elapsed since its last reset. When we use continuous time, there is not always a "last" reset, e.g. when the reset holds in an open interval. For this case, we will use non-standard clock values of the form υ^+, intuitively meaning that the clock was reset υ units before. The set of non-standard real numbers, noted $\mathbb{R}^+_{\geq 0}$, is the set of $\{\upsilon, \upsilon^+ \mid \upsilon \in \mathbb{R}_{\geq 0}\}$, ordered by $<_{ns}$ as following: $\upsilon_1 <_{ns} \upsilon_2^+$ iff $\upsilon_1 \leq \upsilon_2$. The addition is commutative, and $\upsilon_1^+ + \upsilon_2 = (\upsilon_1 + \upsilon_2)^+$. $\mathbb{R}^+_{\perp}$ is $\mathbb{R}^+_{\geq 0}$ plus a special value $\perp$ for uninitialized clocks. $\perp$ is not comparable to other values, and is absorbing for addition.

Let X be a finite set of clock names. A clock valuation over X is a mapping $\nu : X \to \mathbb{R}^+_\perp$. The set of constraints over X, denoted $\Phi(X)$, is defined by the grammar $\phi ::= true \mid x \sim c \mid \phi_1 \wedge \phi_2$ where $x \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, >, \geq\}$. We write $\nu \models \phi$ when the valuation ν satisfies the constraint ϕ. When x has the value $\perp$, we evaluate $x \sim c$ to false.

3 Automata Background

Based on time and clocks, several variants of timed automata have been proposed after the seminal Timed Automata (TA) [2]. Below, we review briefly icTA [1] and RECA [12], that are the basis of our DECA.

We use an interval semantics throughout the paper, i.e. predicates (or letters) are functions of time with finite variability. In particular, we do not allow to be in two locations, or to make two transitions, at the same time. Transitions are taken in a single instant; therefore we have to stay in a location during an open interval. Thus, we have to label both locations and transitions (together, we call them *locansitions*) to ensure that predicates are always defined. Time thus strictly increases along an ISS, as in [2]. We allow unobservable transitions [7], that were absent from [2]: here, they are expressed as a transition with the same label as the previous and next location.

3.1 Timed Automata

A Timed Automaton (TA) [2] is a finite state automaton augmented with clocks: real variables that can be reset to 0, and otherwise increase at a uniform rate. Time is thus global, and clocks are perfectly precise and synchronized.

Definition 1. *A Timed Automaton is a tuple* $\mathcal{A} = (\Sigma, X, S, s_0, \rightarrow_{ta}, Inv, \lambda, F)$, *with:*

- *(i)* Σ, *a finite alphabet. In this paper, we take* $\Sigma = 2^{\mathbb{P}}$.
- *(ii)* X, *a finite set of positive real variables called clocks.*
- *(iii)* S, *a finite set of locations.*
- *(iv)* $S_0 \subseteq S$, *the initial locations.*
- *(v)* $\rightarrow_{ta} \subseteq S \times \Phi(X) \times 2^X \times S$, *a finite set of transitions, each with a guard and a reset.*
- *(vi)* $Inv : S \to \Phi(X)$ *gives the invariant.*
- *(vii)* $\lambda : (S \cup \rightarrow_{ta}) \to \Sigma$, *a labelling of locations and transitions.*
- *(viii)* F, *an acceptance condition. For instance, for finite acceptance, we have* $F \subseteq S$, *a set of final locations. We also use Büchi acceptance (where* $F \subseteq S$*) or parity conditions (where* $F : S \to \mathbb{N}$*).*

TA are neither determinizable nor complementable. Their emptiness problem can be solved using the region construction, but their inclusion problem is undecidable [2].

3.2 Timed Automata with Independent Clocks

Distributed Timed Automata (DTA) [14,1] consist of a number of local timed automata, called processes. Each processes owns clocks. The clocks of a same process evolve synchronously, but independently of the clocks of the other processes. The idea is that the clocks of the same process are all computed from a same hardware clock. A clock can be read by any process, but can only be reset by its owner process.

The homonymous Distributed Timed Automata of [10] work differently: they model processes whose execution is interleaved by a scheduler. Thus, only one process increases its (perfect) clocks at a time. They are a subclass of stopwatch automata.

In [1], DTA are not much studied. Instead, their product is first computed, giving rise to the class of Timed Automata with independent clocks (icTA).

Definition 2. *An* icTA *is a pair* $(\mathcal{A}, \pi)$*, where* $\mathcal{A}$ *is a* TA *and* $\pi : X \to Proc$ *maps each clock to a process.*

Definition 3. *A* Rate *is a tuple* $\tau = (\tau_q)_{q \in Proc}$ *of local time functions. Each local time function* τ_q *maps the reference time to the time of process* q*, i.e,* $\tau_q : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$*. The functions* τ_q *must be continuous, strictly increasing, divergent, and satisfy* $\tau_q(0) = 0$*.*

Note that the reference time is arbitrary, and thus not meaningful.

Definition 4. *Given a clock valuation* $\nu : X \to \mathbb{R}_{\geq 0}$*, a rate* τ*, and two reference times* $t_1 > t_2$*, the valuation* $\nu + (t_1 - t_2)$ *maps* x *to* $\nu(x) + \tau_{\pi(x)}(t_1) - \tau_{\pi(x)}(t_2)$*.*

Definition 5. *A* run *of an* icTA $\mathcal{A}$ *for* τ *is an* ISS *alternating states and transitions* (β, I) *where* $\beta = \zeta_0, q_1, \zeta_1, q_2, \ldots$*,* $I = \{0\},]0, t_1[, \{t_1\},]t_1, t_2[, \ldots$*, the states* $q_i \in Q = \{(s_i, \nu_i) \in S \times \mathbb{R}^X_{\geq 0} \mid \nu_i \models Inv(s_i)\}$*. It must satisfy:*

1. *the starting state is* $q_0 = (s_0, \nu_0)$*, where* ν_0 *assigns* 0 *to all the clocks, and* $s_0 \in S_0$*.*
2. *When spending time* $]t_{i-1}, t_i[$ *in* $q_i = (s_i, \nu_i)$*, the invariant must stay continuously true:* $\forall t \in]t_{i-1}, t_i[: \nu_i + (t - t_{i-1})) \models Inv(s_i)$*.*
3. *When following a transition* $\zeta_i = (s_i, \phi, Y, s_{i+1}) \in \rightarrow_{\mathsf{icTA}}$*, the clock constraint* ϕ *must be satisfied:* $\nu_i + (t_i - t_{i-1}) \models \phi$*. The clocks in* Y *are then reset:* $\nu_{i+1} = (\nu_i + (t_i - t_{i-1}))[Y \to 0]$*. This transition is instantaneous.*
4. *The acceptance condition is verified, e.g. for a finite automaton,* $s_n \in F$*.*

Definition 6. *Given a run* $\rho = (\zeta_0, (s_1, \nu_1), \zeta_1, (s_2, \nu_2), \ldots, I)$ *we define its* ISS*, noted* $\lambda(\rho)$*, as* $(\lambda(\zeta_0), \lambda(s_1), \lambda(\zeta_1), \lambda(s_2), \ldots, I)$*.*

Definition 7. *The language* $\mathcal{L}(\mathcal{B}, \tau)$ *is the set of* ISS *of accepting runs of* $\mathcal{B}$ *for* τ*, closed under* $\equiv$*.*

3.3 Recursive Event Clocks Automata

Recursive Event Clock Automata (RECA) [19,12] extend ECA [5]. "Recursive" refers to the fact that the resets of an event clock $x_{\mathcal{B}}$ are controlled by a lower-level automaton $\mathcal{B}$: When $\mathcal{B}$ visits a monitored locansitions (location or transition), it resets $x_{\mathcal{B}}$. Symmetrically, *prediction clocks* of the form $y_{\mathcal{B}}$ measure the time until $\mathcal{B}$ can next visit one of its monitored locansitions. Distributed Real-Time Automata [9] are a special case of RECA where only the time since the last change of labelling can be measured.

Definition 8. *A RECA $\mathcal{A}$ of level $l \in \mathbb{N}$ is a tuple composed of:*

(i) Σ *is a finite alphabet.*
(ii) S *is a finite set of locations.*
(iii) $S_0 \subseteq S$ *are the initial locations.*
(iv) $\rightarrow_{reca} \subseteq S \times S$ *are the transitions.*
(v) C *is a finite set of clocks, of the form* $x_{\mathcal{B}}$ *or* $y_{\mathcal{B}}$*, with* $\mathcal{B}$ *a lower-level RECA.*
(vi) $\lambda : (S \cup \rightarrow_{reca}) \rightarrow \Sigma$ *is a labelling function.*
(vii) $\mathsf{Inv} : (S \cup \rightarrow_{reca}) \rightarrow \Phi(C)$ *gives the guard or invariant.*
(viii) $M \subseteq (S \cup \rightarrow_{reca})$ *is the set of monitored locansitions: when the automaton visits them, it resets its two associated clocks* $x_{\mathcal{A}}, y_{\mathcal{A}}$*.*
(ix) F *is an acceptance condition.*

RECA can be determinized and thus complemented: They are *fully decidable* [19,12]. They are quite expressive, since they can express the logic MITL [3], but less expressive than TA (otherwise we would lose full decidability).

Below, we assume the uniform naming conventions defined in this section.

4 Timed Languages

Surprisingly, Akshay et al. [1] only consider untimed languages for their timed automata. We are interested in timed languages, but we have a different time scale for each process; thus each process p will determine a timed language observed by p. These languages only differ by their timings. Let τ_p be the rate of process p. τ_p extends naturally to intervals, to interval sequences, to ISS, and to timed languages: Given an ISS $\theta = (\sigma, I)$ expressed in the reference time, $\tau_p(\theta)$ is $(\sigma, \tau_p(I))$. The timed language for τ observed by p is $\tau_p(\mathcal{L}(\mathcal{B}, \tau))$. When there is only one process, i.e. $Proc = \{q\}$, the timed language observed by q is the usual timed language $\mathcal{L}(\mathcal{A})$ of its TA. When τ is a vector of identity functions, we also obtain the usual timed language whatever the observer process chosen.

When we want to avoid some forbidden timed behaviours (ISS) , we consult the existential timed semantics: we consider time evolutions as non-deterministic, since this gives the worst-case assumption. If we want a given timed behaviour to be possible whatever the evolution of local times, we look at the universal semantics.

Definition 9. *For an automaton $\mathcal{B}$ and one of its processes p, we define:*

- *the existential timed language observed by* $p : \mathcal{L}_\exists(\mathcal{B}, p) = \bigcup_{\tau \in Rates} \tau_p(\mathcal{L}(\mathcal{B}, \tau))$
- *the universal timed language observed by* $p : \mathcal{L}_\forall(\mathcal{B}, p) = \bigcap_{\tau \in Rates} \tau_p(\mathcal{L}(\mathcal{B}, \tau))$

This variety of languages leads to three generalisations of the classical problems of emptiness, inclusion, intersection and union. First, the τ-wise definitions:

Definition 10. *Given icTA* $\mathcal{A}, \mathcal{B}, \mathcal{C}$*,*

1. $\mathcal{C}$ *is an intersection of* $\mathcal{A}, \mathcal{B}$ *iff* $\forall \tau \in Rates, \mathcal{L}(\mathcal{C}, \tau) = \mathcal{L}(\mathcal{A}, \tau) \cap \mathcal{L}(\mathcal{B}, \tau)$
2. $\mathcal{C}$ *is a union of* $\mathcal{A}, \mathcal{B}$ *iff* $\forall \tau \in Rates, \mathcal{L}(\mathcal{C}, \tau) = \mathcal{L}(\mathcal{A}, \tau) \cup \mathcal{L}(\mathcal{B}, \tau)$
3. $\mathcal{C}$ *is a complement automaton of* $\mathcal{A}$ *iff* $\forall \tau \in Rates, \mathcal{L}(\mathcal{C}, \tau) = \mathcal{L}(\mathcal{A}, \tau)^c$*, where* c *is the complement operator.*
4. $\mathcal{A}$ *is language-included in* $\mathcal{B}$ *iff* $\forall \tau \in Rates, \mathcal{L}(\mathcal{A}, \tau) \subseteq \mathcal{L}(\mathcal{B}, \tau)$
5. *The emptiness problem for* $\mathcal{A}$ *is* $\forall \tau \in Rates, \mathcal{L}(\mathcal{A}, \tau) = \emptyset$

The p-existential and p-universal variants use respectively the existential and universal timed languages observed by p.

4.1 Timed Languages of icTA

The existential timed languages of icTA are *timed regular* [2]:

Theorem 1. *For any icTA* $\mathcal{B}$*,* $\mathcal{L}_\exists(\mathcal{B}, p)$ *is the language of a TA.*

Proof. (sketch) This TA can be computed by a variant of the region construction, of which the construction of [1] is a special case. Let $q \in Proc \setminus \{p\}$ be a process whose clocks we want to eliminate, i.e. we have an icTA $\mathcal{B}$ on $Proc$ and we would like to construct an icTA on $Proc \setminus \{q\}$ whose existential language is preserved for any observer but q. We construct the region automaton, but on the clocks of q only. If a locansition had invariant $\bigwedge_{p \in Proc} \phi_p$, its associated regions have invariant $\bigwedge_{p \in Proc\{q\}} \phi_p$. The constraints on clocks of q are not lost, they become part of the region constraint. This gives a region icTA without the clocks of q, and where the locations are now a pair of an original location and a region constraint on clocks of q, which has the required languages. If we want to eliminate several processes, we eliminate them one by one: eliminating several processes together would give a result that does not reflect the independence of their clocks.

However, the emptiness of their universal timed languages is undecidable, and thus cannot be the language of a TA.

Theorem 2. *icTA are closed under* τ*-wise and* p*-existential intersection and union, and under* p*-universal intersection.*

However, icTA are not determinizable, nor closed under timed complement, and their inclusion problem is undecidable (whether τ-wise, p-existential, or p-universal), essentially because TA [2] are a special case of icTA.

5 Distributed Event Clock Automata

To restore full decidability, we use event clocks [5]. For expressiveness, we use RECA [12] with independent clocks [1]. The distributed event clock (DEC) $x^q_{\mathcal{A}}$ (or $y^q_{\mathcal{A}}$) records the time since the last (resp. next) time that the automaton $\mathcal{A}$ could visit a monitored locansition, measured in the local time of process q.

Definition 11. *A distributed recursive event clock automaton (DECA) is a pair* $(\mathcal{A}, \pi)$ *where* $\mathcal{A}$ *is a RECA and* $\pi : C \to Proc$ *maps each clock to a process.*

For better readability, we write the owner process in the clock name: $\pi(x^q_{\mathcal{A}}) = q$.

Definition 12. *A* run ρ *of a DECA* $\mathcal{A}$ *for a rate* τ *is an ISS alternating transitions and locations* $(\zeta_0, s_1, \zeta_1, s_2, \ldots, I)$*, such that:*

(i) The run starts from an initial location: $\zeta_0 \in S_0 \times S$.
(ii) The run follows discrete transitions: $\zeta_i = (s_i, s_{i+1}) \in \rightarrow_{reca}$
(iii) The clock constraints (invariant or guard) are satisfied by the valuation of the clocks (defined below): $\forall t \in \mathbb{R}_{\geq 0}, \nu(\lambda(\rho), t, \tau) \models Inv(\rho(t))$.
(iv) It satisfies the acceptance condition.

Definition 13. *The ISS of a run* $\rho = (s, I)$*, noted* $\lambda(\rho)$*, is the pair* $(\lambda(s), I)$.

Definition 14. $\mathcal{A}$ *accepts an ISS* θ *at* t *with* τ*, if there is a run* ρ *for* θ *that visits a monitored location at* t*. This is noted* $(t, \theta) \in \mathcal{L}^+(\mathcal{A},\tau)$*, its* anchored language.

This acceptance time will be used to reset the associated clocks $x^q_{\mathcal{A}}$ below.

Definition 15. *The DEC* valuation *depends on the ISS* θ*, on the reference time of evaluation* t*, and on the rate* τ*. It assigns a (non-standard) positive real, or undefined, to each clock variable.*

$$\nu(\theta, t, \tau, x^q_{\mathcal{B}}) = \begin{cases} \tau_q(t) - \tau_q(r) & \text{if } r = \max\{s < t | (s, \theta) \in \mathcal{L}^+(\mathcal{B},\tau)\} \text{ exists} \\ (\tau_q(t) - \tau_q(r))^+ & \text{else, if } r = \sup\{s < t | (s, \theta) \in \mathcal{L}^+(\mathcal{B},\tau)\} \text{ exists} \\ \bot & \text{else} \end{cases}$$

$$\nu(\theta, t, \tau, y^q_{\mathcal{B}}) = \begin{cases} \tau_q(l) - \tau_q(t) & \text{if } l = \min\{s > t | (s, \theta) \in \mathcal{L}^+(\mathcal{B},\tau)\} \text{ exists} \\ (\tau_q(l) - \tau_q(t))^+ & \text{else, if } l = \inf\{s > t \mid (s, \theta) \in \mathcal{L}^+(\mathcal{B},\tau)\} \text{ exists} \\ \bot & \text{else} \end{cases}$$

Definition 16. *The timed language of a DECA* $\mathcal{A}$*, noted* $\mathcal{L}(\mathcal{A}, \tau)$*, are the ISS of its runs for* τ*, closed under* $\equiv$.

Example 1. The example of Fig.1 from [1] is in fact both a DECA and an icTA $\mathcal{A}$ over $Proc = \{p, q\}$, and the set of propositions $\mathbb{P} = \{a, b, c\}$. Locations have no invariant and an ϵ labelling. Both clocks are reset by the initial monitored transition of $\mathcal{B}$. After this, they may diverge. The existential timed languages, here, are read from the automaton:

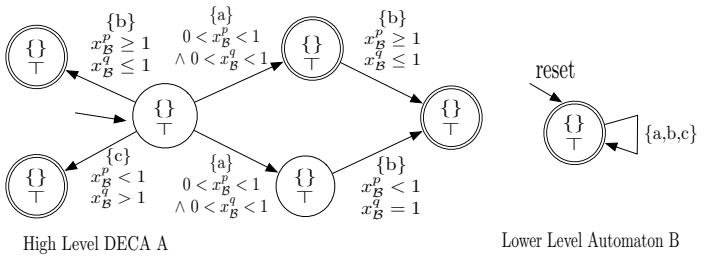

Fig. 1. Example of DECA from [1]

$$\mathcal{L}_\exists(\mathcal{A},p) = \mathsf{ITL}^1(\{(a,t_1^p) \mid 0 < t_1^p < 1\} \cup \{(b,t_1^p) \mid t_1^p \geq 1\} \cup \{(c,t_1^p) \mid 0 < t_1^p < 1\} \\ \cup\{(a,t_1^p),(b,t_2^p) \mid 0 < t_1^p < 1 \wedge t_1^p < t_2^p\})$$
$$\mathcal{L}_\exists(\mathcal{A},q) = \mathsf{ITL}(\{(a,t_1^q) \mid 0 < t_1^q < 1\} \cup \{(b,t_1^q) \mid 0 < t_1^q \leq 1\} \cup \{(c,t_1^q) \mid t_1^q > 1\} \\ \cup\{(a,t_1^q),(b,t_2^q) \mid 0 < t_1^q < 1 \wedge t_1^q < t_2^q \leq 1\})$$

Here, all universal timed languages are empty: $\mathcal{L}_\forall(\mathcal{A},p) = \emptyset = \mathcal{L}_\forall(\mathcal{A},q)$. For instance, we cannot have $(a,t_a) \in \mathcal{L}_\forall(\mathcal{A},p)$, because there are some τ where the time of q increases steeply, and gets over 1 before the time of p could reach t_a. However, the universal untimed language $\mathcal{L}_\forall(\mathcal{A})$ is $\{a, ab\}$.

5.1 Timed Languages of DECA

DECA inherit the main property of RECA: they are determinizable. The theorems below are valid for the finite version, but also for the infinite ones, e.g. for Büchi automata, which are determinized to a parity automaton [17].

Definition 17. *A DECA $\mathcal{A}$ is deterministic iff all the following conditions hold:*

(i) *$\mathcal{A}$ has exactly one initial location $\{s_0\} = S_0$;*
(ii) *It has no ϵ-transitions: There are no two successive locations $s_1 \to s_2$, with the same labellings: $\lambda(s_1) = \lambda(s_1,s_2) = \lambda(s_2)$.*
(iii) *Any two distinct successor locations $s_2 \neq s_3, s_1 \to s_2, s_1 \to s_3$ with same labellings: $\lambda(s_2) = \lambda(s_3)$ and $\lambda(s_1,s_2) = \lambda(s_1,s_3)$, have mutually exclusive clock constraints: $\nu \not\models \mathit{Inv}(s_1,s_2) \wedge \mathit{Inv}(s_1,s_3)$.*

Theorem 3. *Given a deterministic DECA, a rate τ, an ISS θ, there is at most one accepting run on τ for θ, i.e. $\lambda(\rho) \equiv \theta$.*

As for RECA, prediction clocks render a deterministic DECA dependent on the future, and thus unsuitable for realizability [11].

We don't have space to present our determinization construction [19], but its complications rather stems from continuous time than from DEC.

[1] ITL will add the missing intervals between time points.

Theorem 4. *Determinization preserves the τ-wise, existential and universal languages.*

Theorem 5. *DECA are closed under union, intersection and complementation, whether τ-wise, p-existential and p-universal.*

Theorem 6. *For all DECA $\mathcal{B}, p \in Proc, \mathcal{L}_\exists(\mathcal{B}, p)$ is the language of a RECA.*

Proof. (sketch) We eliminate each process $q \neq p$ in turn from the DECA while preserving the existential language of the remaining processes. We first complete and determinize automata appearing in the clocks of this process q. We then make their product with the main automaton. We then perform the region construction [3] on the clocks of q. Remember that the clocks are constrained to be 0 in the respective monitored locansition, i.e. when at least one original monitored locansition appears in this construction, and that prediction clocks run backwards so that it is the complement of their fractional part that participates in the region construction [3]. The region construction for prediction clocks is non-deterministic and is not a bisimulation quotient, unlike the one of TA, but preserves the language [19]. Note that the elimination of the clocks of one process only, allows independent evolution of the other clocks.

Theorem 7. *The τ-wise and p-existential emptiness, universality and language inclusion problem for DECA are PSPACE-complete.*

Finite automata have the same complexity, thus the added expressiveness is "for free".

Theorem 8. *For all DECA $\mathcal{B}, q \in Proc, \mathcal{L}_\forall(\mathcal{B}, q)$ is the language of a RECA.*

Proof. (sketch) We complete and determinize the main automaton $\mathcal{A}$. Then we apply the region construction for independent clocks [1]. The automaton becomes non-deterministic, because each region has several successors, depending on τ. Transitions are also considered as regions. A region constraint is expressed as a conjunction $\bigwedge_{p \in Proc} \phi_p$. We choose as invariant of each region locansition ϕ_q. The other constraints are part of the identity of the region, but are not kept as an invariant. Then we determinize it again but we mark as final the locations where all members are final (which, in turn, means that one of their members is an original final location), to represent that the ISS must be accepted under *all* evolutions of time τ.

In contrast, the universal language of DTA and icTA is undecidable [1].

6 Recursive Distributed Event Clocks Temporal Logic

The aim of this section is to construct a fully decidable distributed logic to specify real-time requirements when time scales can be independent.

6.1 Syntax

Our Distributed Event Clock Temporal Logic (DECTL) extend the Event Clock Temporal Logic (EventClockTL) [20,12] with distributed (a.k.a. independent) real-time modalities. As in Section 3.2, we assume a set of processes *Proc*. The clocks of each process will evolve according to its local time given by a Rate τ.

DECTL is based on LTL, and adds two local real-time modalities. The recording modality $\lhd_I^q \phi$ means that ϕ was true for the last time at reference time t_1 and that the distance, as measured by the time scale of q, is within the interval I: $\tau_q(t_0) - \tau_q(t_1) \in I$. Symmetrically, the predicting modality $\rhd_I^q \phi$ says that the next ϕ will occur within I according to the local time of q. With only one process, we find back EventClockTL [20].

Definition 18. *The* formulas *of DECTL are defined by the grammar:*

$$\phi ::= \ true \mid p \mid \ \rhd_I^q \ \phi \mid \ \lhd_I^q \ \phi \mid \phi_1 \ \wedge \ \phi_2 \mid \neg\phi \mid \phi_1 \, \mathcal{U} \, \phi_2 \mid \phi_1 \, \mathcal{S} \, \phi_2$$

where $p \in \mathbb{P}$ is an atom, $I \in \mathcal{I}(\mathbb{N})$ is an interval and $q \in Proc$ is a process.

6.2 Semantics

Definition 19. *The* satisfaction *of a DECTL formula ϕ is noted $(t, \theta) \models_\tau \phi$. We omit τ and θ below, since they are fixed.*

$$\begin{array}{lll}
t \models & p & \text{iff } p \in \theta(t) \\
t \models & \neg\phi & \text{iff } t \not\models \phi \\
t \models & \phi_1 \wedge \phi_2 & \text{iff } t \models \phi_1 \text{ and } t \models \phi_2 \\
t \models & \phi_1 \mathcal{U} \phi_2 & \text{iff } \exists t' > t, t' \models \phi_2 \text{ and } \forall t'' \in (t, t'), t'' \models \phi_1 \\
t \models & \phi_1 \mathcal{S} \phi_2 & \text{iff } \exists t' < t, t' \models \phi_2 \text{ and } \forall t'' \in (t', t), t'' \models \phi_1 \\
t \models & \lhd_I^q \phi & \text{iff } \exists t' < t, \tau_q(t) - \tau_q(t') \in I \wedge t' \models \phi \\
 & & \text{and } \forall t'' < t, \tau_q(t) - \tau_q(t'') < I, t'' \not\models \phi \\
t \models & \rhd_I^q \phi & \text{iff } \exists t' > t, \tau_q(t') - \tau_q(t) \in I \wedge t' \models \phi \\
 & & \text{and } \forall t'' > t, \tau_q(t'') - \tau_q(t) < I, t'' \not\models \phi
\end{array}$$

Example 2. The formula $\neg(\mathcal{F}b \wedge \neg \rhd_{\leq 1}^q b)$, where $\mathcal{F}b = true \, \mathcal{U} b$ says that the first b, if any, must occur within 1 second, as measured by q. It holds on the automation of Fig.1. However, the formula measured by p, $\neg(\mathcal{F}b \wedge \neg \rhd_{\leq 1}^p b)$, does not hold.

6.3 Decidability

Theorem 9. *For any DECTL formula ϕ, there is a DECA automaton $\mathcal{A}_\phi$ with the same anchored language: $(t, \theta) \in \mathcal{L}^+(\mathcal{A}, \tau)$ iff $(t, \theta) \models_\tau \phi$.*

Proof. (sketch) The translation to a Generalised Büchi tableau is done level by level, where the level of a formula is the nesting depth of real-time modalities [19]. A formula $\rhd_I^q \phi$ is translated as constraint $x_{\mathcal{A}_\phi}^q \in I$. The monitored locations of $\mathcal{A}_\phi$ are those containing ϕ. The initial locations are those containing

$\neg true\ \mathcal{S} true$. The transitions are the sets of closure formulae that entail instantaneity. Each location has the Hintikka property: the conjunction of its formulae is satisfied exactly by the ISS of the runs visiting it, at the time they visit it.

The construction is exponential in the size of the non-real time part of the formula, but linear in the real-time part. The test of emptiness is done by the region construction presented in Section 5, that is exponential in the real-time part but linear for the rest.

Theorem 10. *Satisfiability and validity of DECTL are PSPACE-complete.*

The axiomatisation of this logic happens to be given in [21]. There, *shift and order axioms* express the pairwise synchronisation of real-time modalities. We restrict them to modalities of the same process.

6.4 Extensions

(1) We can extend the known expressive equivalence of EventClockTL and MITL+ Past [12] to construct a distributed version of MITL (DMITL) with independent modalities $\mathcal{U}_I^p, \mathcal{S}_I^p$.
(2) DECTL is expressively equivalent to DQTL, a new first-order monadic logic with a metric quantifiers $\exists t \in^p] t_0, t_0 + k[\ .\ \phi,\ \exists t \in^p] t_0 - k, t_0[\ .\ \phi$, where ϕ has only the free variable t (see [13] for QTL).
(3) The more expressive logic DMECTL allows to observe not only the last ϕ, but also the last but n ϕ [15]. This logic is still translatable in DECA.
(4) This logic is expressively equivalent to DQ2MLO, a new first-order monadic logic with a metric quantifier $\exists t \in^p] t_0, t_0 + k[\ .\ \phi,\ \exists t \in^p] t_0 - k, t_0[\ .\ \phi$, where ϕ has only the free variables t_0, t (see [13] for Q2MLO).
(5) We can add DECA automata operators [22].
(6) We can add second-order quantification on predicates that are not subjected to a real-time constraint.
(7) We can also introduce these independent modalities $\mathcal{U}_I^p, \mathcal{S}_I^p$ in a linear μ-calculus.

The last three extensions are expressively equivalent.

7 Conclusions

We have proposed the basis of a framework for analyzing distributed real-time systems through of the introduction of independent (or distributed) event clocks, inspired by icTA [1]. In contrast to [1], we have given a real-time semantics, and thus we can specify real-time properties. We have defined DECA and showed that they are fully decidable, and that their language inclusion problem is PSPACE-complete, as for classical automata. This give us an algorithm to verify real-time properties. Since the number of regions is reduced wrt. ECA, we can even expect faster verification. They are also a good basis for partial-order techniques [6]. The universal (timed) languages of DECA are decidable and (timed) regular, unlike

the universal languages of icTA [1]. We proposed the logic DECTL to specify real-time properties with distributed observers. The problems of satisfiability, validity and model-checking of DECTL are PSPACE-complete, as for LTL - we cannot hope better.

Acknowledgements

This work was funded by the Interuniversity Attraction Poles Programme (IAP) of the Belgian State, Belgian Science Policy (MoVES project), by the Belgian Science Foundation (FNRS) under FRFC project CFV, by the Hubert Curien Grants (Tournesol), and by the European Science Foundation (ESF) under EUROCORES project LogiCCC/GASICS.

References

1. Akshay, S., Bollig, B., Gastin, P., Mukund, M., Narayan Kumar, K.: Distributed timed automata with independently evolving clocks. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 82–97. Springer, Heidelberg (2008)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
3. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. ACM 43(1), 116–146 (1996)
4. Alur, R., Fix, L., Henzinger, T.A.: A determinizable class of timed automata. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 1–13. Springer, Heidelberg (1994)
5. Alur, R., Henzinger, T.A.: Logics and models of real time: A survey. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 74–106. Springer, Heidelberg (1992)
6. Bengtsson, J.E., Jonsson, B., Lilius, J., Yi, W.: Partial order reductions for timed systems. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 485–500. Springer, Heidelberg (1998)
7. Bérard, B., Petit, A., Diekert, V., Gastin, P.: Characterization of the expressive power of silent transitions in timed automata. Fundam. Inform. 36(2-3), 145–182 (1998)
8. De Wulf, M., Doyen, L., Markey, N., Raskin, J.-F.: Robustness and implementability of timed automata. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 118–133. Springer, Heidelberg (2004)
9. Dima, C.: Distributed real-time automata. In: Martin-Vide, C., Mitrana, V. (eds.) Essays in honor of Gheorghe Păun, pp. 131–140. Taylor & Francis, Abington (2003)
10. Dima, C., Lanotte, R.: Distributed time-asynchronous automata. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) ICTAC 2007. LNCS, vol. 4711, pp. 185–200. Springer, Heidelberg (2007)
11. Doyen, L., Geeraerts, G., Raskin, J.-F., Reichert, J.: Realizability of real-time logics. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 133–148. Springer, Heidelberg (2009)
12. Henzinger, T.A., Raskin, J.-F., Schobbens, P.-Y.: The regular real-time languages. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 580–591. Springer, Heidelberg (1998)

13. Hirshfeld, Y., Rabinovich, A.: An expressive temporal logic for real time. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 492–504. Springer, Heidelberg (2006)
14. Krishnan, P.: Distributed timed automata. Electr. Notes Theor. Comput. Sci. 28 (1999)
15. Jerson Ortiz, J., Legay, A., Schobbens, P.-Y.: Memory event clocks. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 198–212. Springer, Heidelberg (2010)
16. Ortiz, J., Legay, A., Schobbens, P.Y.: Distributed event clock automata. Tech. rep., FUNDP University, Belgium (2011), `http://www.info.fundp.ac.be/$\sim$jor/DecaReport`
17. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science, pp. 255–264. IEEE Computer Society, Washington, DC, USA (2006), `http://portal.acm.org/citation.cfm?id=1157735.1158062`
18. Puri, A.: Dynamical properties of timed automata. In: Ravn, A.P., Rischel, H. (eds.) FTRTFT 1998. LNCS, vol. 1486, pp. 210–227. Springer, Heidelberg (1998)
19. Raskin, J.F.: Logics, Automata and Classical Theories for Deciding Real Time. Phd thesis, FUNDP University, Belgium (1999), `http://www.ulb.ac.be/di/ssd/jfr/`
20. Raskin, J.F., Schobbens, P.Y.: State clock logic: A decidable real-time logic. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 33–47. Springer, Heidelberg (1997)
21. Schobbens, P.Y., Raskin, J.F., Henzinger, T.A.: Axioms for real-time logics. Theoretical Computer Science 274, 151–182 (2002)
22. Wolper, P.: Temporal logic can be more expressive. Information and Control 56(1-2), 72–99 (1983)