Memory Event Clocks

Ortiz, James; Legay, Axel; Schobbens, Pierre-Yves

Published in:
Formal Modeling and Analysis of Timed Systems

DOI:
10.1007/978-3-642-15297-9_16

Publication date:
2010

Document Version
Early version, also known as pre-print

PReCISE – FUNDP
University of Namur
Rue Grandgagnage, 21
B-5000 Namur
Belgium

PAPER March, 2011

AUTHORS	J. Ortiz, A. Legay, P-Y. Schobbens
EMAIL	jor@info.fundp.ac.be, alegay@irisia.fr, pierre-yves.schobbens@fundp.ac.be
VENUE	FORMATS'10
STATUS	Extended version of the paper accepted in the proceedings of the 8th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'10), IST Austria, Klosterneuburg, Austria
PROJECT FUNDING	MoVES Interuniversity Attraction Poles Programme (IAP) of the Belgian State, Belgian Science Policy (MoVES project), by the Belgian Science Foundation (FNRS) under FRFC project CFV, and by the European Science Foundation (ESF) under EUROCORES project LogiCCC/GASICS

Memory Event Clocks

Memory Event Clocks
Extended Abstract

James Ortiz[1], 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. We introduce logics and automata based on memory event clocks. A memory clock is not really reset: instead, a new clock is created, while the old one is still accessible by indexing. We can thus constrain not only the time since the last reset (which was the main limitation in event clocks), but also since previous resets. When we introduce these clocks in the linear temporal logic of the reals, we create Recursive Memory Event Clocks Temporal Logic (RMECTL). It turns out to have the same expressiveness as the Temporal Logic with Counting (TLC) of Hirshfeld and Rabinovich. We then examine automata with recursive memory event clocks (RMECA). Recursive event clocks are reset by simpler RMECA, hence the name "recursive". In contrast, we show that for RMECA, memory clocks do not add expressiveness, but only concision. The original RECA define thus a fully decidable, robust and expressive level of real-time expressiveness.

1 Introduction

Finite automata is a widely used computational model to capture and analyse the behavior of possibly concurrent systems. The main question is checking whether an automaton satisfies a given specification, which can be represented either by some temporal logic formula or by another automaton. The first case, called model checking, is usually reduced to the second. In the second case, the problem is called language inclusion between automata, which models step-wise refinement.

Nowadays, real-time plays a crucial role in system design, especially in the area of embedded systems. To capture the behavior of a real-time system, one needs to augment the computational model with a notion of time. An important model is timed automata (TA) [1], that are automata augmented with clocks used to monitor the evolution of time. Timed automata offer tools [21, 9, 6] for many real-time problems. Unfortunately, TA have an undecidable language inclusion problem [1]. Around the same time, the satisfiability of natural real-time logics such as Metric Temporal Logic (MTL) and Temporal Propositional Timed Logic (TPTL) were also proved undecidable [5]. In fact, one of the central problems is that TA are not closed under determinization (see [16, 15, 7] for discussions). 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). When all these problems are decidable, we call the formalism (automata or logic) fully decidable. These negative results spurred a quest for expressive but still fully decidable formalisms.

To overcome the problem, [3] proposed to restrict the behavior of TA clocks in such a way that language inclusion becomes decidable. The key idea is that the problematic clocks of TA are reset by non-deterministic, internal transitions, that prevent determinization. In contrast, an event clock (EC) x_p is reset when the atomic proposition p occurs. The event clock resets and values are determined by the input and thus Event Clock Automata (ECA) are determinizable, making language inclusion decidable and thus enabling step-wise refinement.

Event clocks can also be introduced in temporal logics [20]. An event clock constraint is naturally translated into a proposition $\lhd_I p$, that means "the last time that a p occurred was a time d ago, where d lies in I". However, the expressiveness of ECA is rather weak. Indeed, events are just the last or next occurrence of an atomic proposition. For instance, the property "p is continuously true in interval $(0, 1)$" cannot be expressed by such an event clock formula: any model where the distances between p's are below 1 will see the clocks always below 1, whether or not it satisfies this property. Therefore [11] 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. For instance, the property above can be expressed as $\neg \rhd_{(0,1)} \neg p$: $\rhd_I$ is now a modality that can contain any subformula, and can be nested. The temporal logic of recursive event clocks (variously called SCL [20] or EventClockTL [11]) has the same expressiveness as Metric Interval Temporal Logic MITL [2] (a decidable fragment of MTL where punctual constraints are forbidden) in the interval semantics. First-and second-order monadic logics with matching expressiveness have been provided [11], yielding a natural, robust, fully decidable level of real-time expressiveness. However, the expressiveness of event clock models has still been criticized, because event clocks can only constrain the time since the last (or next) event. For instance, EventClockTL cannot express the assumption that no more than 3 requests per second will arrive, or the requirement that these all requests will be treated within the next second, when the treatment requires several steps.

In this paper, we address the above limitation and introduce memory clocks, already sketched in [3]. A memory clock x is not really reset: instead, a new clock is created, while the old one is still accessible by indexing: x^1 will be the usual value of the clock x, i.e. the time since last reset, while x^2 will be the time since the last but one reset. In general, x^i will be the time since the last but i reset. Said otherwise, a reset will save a copy of the clock of index i in the clock of index $i + 1$. It can be seen as a series of clock updates: $x^3 := x^2; x^2 := x^1; x^1 := 0$. Here, we will study the recursive variant, as explained above.

Our first contribution is to extend the EventClockTL logic with memory clocks. This gives us the memory event clocks logic (RMECTL), which we show to be PSPACE-complete if the indices of the clocks are encoded in unary and EXPSPACE-complete for the binary case. RMECTL is strictly more expressive than EventClockTL. To ob-

tain these results, we show that the expressiveness of RMECTL is equivalent to the one of the Temporal Logic with Counting (TLC) [13]. It is worth observing that TLC was inspired by a TPTL formula (see Section 3.3). TPTL is a "really temporal" but highly undecidable logic [5]. We isolate a decidable fragment of TPTL, which we call TPTL1R. It also has the same expressive power, showing the robustness of this level of expressiveness. Our second contribution is to extend RECA with memory clocks. Surprisingly, RMECA are as expressive as the original Recursive Event Clock Automata RECA [11]. However, in the binary case, they may be exponentially more succinct.

Structure of the paper. The rest of the paper is organized as follows. Sections 2 recalls preliminary notions. Section 3 examines real-time temporal logics. First, it recalls TLC [18], then introduce RMECTL and shows its equivalence with TLC. Then, it defines a fragment of TPTL that also has the same expressiveness. Section 4 defines Recursive Memory Event Clock Automata (RMECA), studies their properties, and concludes that they can be reduced to good old RECA [11].

2 Preliminaries

We briefly recall the various models of time that are used in the literature [4]. 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. Classical automata and LTL also use a linear discrete model of time. The *point semantics* adds a time stamp to each event of this discrete model.

Our goal here is to model real-time reactive systems, and thus we will use the real numbers as our model of time. This avoid a premature commitment to a discretization of time: even if computer systems are often discrete, their discretization grain (e.g. clock speed) should not appear at requirements level.

Let $\mathbb{P}$ be a set of propositional symbols. A state over $\mathbb{P}$ is an element of $2^{\mathbb{P}}$. Let $\mathbb{N}$ the set of nonnegative integers, $\mathbb{R}$ denote the set of reals, $\mathbb{R}^+$ the set of nonnegative reals.

In this paper, we use the *interval semantics*. An interval is a convex subset of $\mathbb{R}^+$. An interval is *singular* if it is a singleton. Two intervals I and I' are said to be *adjacent* when $I \cap I' = \emptyset$ and $I \cup I'$ is an interval. We denote by $\mathcal{I}_{\mathbb{R}^+}$ the set of intervals whose bounds are in $\mathbb{R}^+$. An interval sequence over $\mathbb{R}^+$ is an infinite sequence $I = I_0 I_1 \cdots$ of non-empty intervals of $\mathcal{I}_{\mathbb{R}^+}$ where

1. successive intervals I_j and I_{j+1} are adjacent and $I_j < I_{j+1}$, for all $j \geq 0$
2. I is *covering*, i.e., for every $t \in \mathbb{R}^+$, there exists $j \in \mathbb{N}$ such that $t \in I_j$.

An interval state sequence (ISS) is a pair $\rho = (\sigma, I)$ where $\sigma = \sigma_0 \sigma_1 \cdots$ is an infinite sequence of states and $I = I_0 I_1 \cdots$ is an *interval sequence*. A interval state sequence ρ can equivalently be seen as an sequence of elements in $2^{\mathbb{P}} \times \mathcal{I}_{\mathbb{R}^+}$. It can also

be seen as a *signal*, i.e. a function from $\mathbb{R}^+$ to states: Let $\rho = (\sigma, I)$ be a interval state sequence and given $t \in \mathbb{R}^+$, let $i \in \mathbb{N}$ be the interval such that $t \in I_i$. We define $\rho(t)$ as the state σ_i. A signal derived from an ISS will always have finite variability. Below, our automata will consider two ISS that define the same signal as equivalent, even if the intervals might be split differently. Our automata assume finite variability. In contrast, our logics will admit infinite variability.

Given two intervals I_1, I_2, we define the interval between I_1 and I_2 by $BetwI(I_1, I_2) = \{x \mid I_1 < x < I_2\}$.

Given a set S and an interval I, we define S *Begins During* I by $\exists t \in (S \cap I)$, and $\nexists t' \in S$ such that $t' < I$. Symmetrically, we define S *Ends During* I iff $\exists\, t, t \in (S \cap I)$, and $\nexists\, t' \in S$ such that $t' > I$.

2.2 Clocks

The value of a clock is the time elapsed since its last reset. When we use real numbers, there is not always a "last" reset but just a limit, e.g. when the reset holds in an open interval. For this case, we will use non-standard clock values of the form v^+. The set of non-standard reals, noted $\mathbb{R}^+_{ns}$, is the set of $\{v, v^+ \mid v \in \mathbb{R}^+\}$, ordered by $<_{ns}$ as following: $v_1 <_{ns} v_2^+$ iff $v_1 \leq v_2$. $\mathbb{R}^+_\perp$ is $\mathbb{R}^+_{ns}$ plus a special value $\perp$ for uninitialized clocks. $\perp$ is not comparable to other values.

Let X be a finite set of clock names. A clock valuation over X is a mapping $\nu : X \to \mathbb{R}^+_\perp$. The *constraints* over X, noted $\Phi(X)$, are defined by the following grammar, where ϕ ranges over $\Phi(X)$, $x \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, >, \geq\}$:

$$\phi ::= true \mid x \sim c \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi$$

We write $\nu \models \phi$ when the valuation ν satisfies the constraint ϕ. By convention, the value $\perp$ does not satisfy any constraint except $true$.

3 Temporal Logics

As we said in the Introduction, the goal of our quest is to construct two levels of expressiveness: (i) fully decidable real-time logics to specify requirements on systems; we examine them in this section; (ii) fully decidable real-time automata, that can express these logics, and model systems (see Section 4). In this way, specifications and systems can be handled uniformly, and verification can be automated. We first recall Temporal Logic with Counting (TLC) [13], since our new logics will turn out have the same expressiveness. Then we define the new logics: RMECTL, that includes memory event clocks, and TPTL1R, a decidable fragment of the logic TPTL with the same expressiveness.

3.1 Temporal Logic with Counting

The Temporal Logic with Counting [13] is an extension of the Temporal Logic of the Reals with Past by *counting modalities*. Here we use a slight variant, called $TLCI_0$ [18], where the counting modalities are $C_k^{(0,b)}(\phi)$ and $\overleftarrow{C}_k^{(0,b)}(\phi)$. The modality $C_k^{(0,b)}(\phi)$

says that ϕ will be true at least at k points in the interval $(t, t+b)$, and its symmetrical $\overleftarrow{C}_k^{(0,b)}(\phi)$ says that ϕ has happened k times in the interval $(t-b, t)$. The syntax of $TLCI_0$ formulae is given by:

$$\phi ::= \ p \mid \phi_1 \ \wedge \ \phi_2 \mid \neg \, \phi \mid \phi_1 \, \mathcal{U} \, \phi_2 \mid \phi_1 \, \mathcal{S} \, \phi_2 \mid C_k^{(0,b)}(\phi) \mid \overleftarrow{C}_k^{(0,b)}(\phi)$$

where p is a propositional symbol, $k \in \mathbb{N}, b \in \mathbb{N}$ and $\phi_1, \phi_2 \in TLCI_0$.

Formally, the semantics is as follows:

$$(\rho, t_0) \models C_k^{(0,b)}(\phi) \ \text{ iff } \ \exists t_1 \cdots \exists t_k, t_0 < t_1 < \cdots < t_k < t_0 + b \ \ \wedge \bigwedge_{0<i\leq k} \phi(t_i)$$

$$(\rho, t_0) \models \overleftarrow{C}_k^{(0,b)}(\phi) \ \text{ iff } \ \exists t_1 \cdots \exists t_k, t_0 - b < t_1 < \cdots < t_k < t_0 \ \wedge \bigwedge_{0<i\leq k} \phi(t_i)$$

Let us also recall the classical semantics, that we will use throughout the paper:

$$\begin{array}{lll}
(\rho,t) \models & p & \text{iff } p \in \rho(t) \\
(\rho,t) \models & \neg\phi & \text{iff } (\rho,t) \not\models \phi \\
(\rho,t) \models \phi_1 \wedge \phi_2 & & \text{iff } (\rho,t) \models \phi_1 \text{ and } (\rho,t) \models \phi_2 \\
(\rho,t) \models \phi_1 \mathcal{U} \phi_2 & & \text{iff } \exists t' > t. (\rho,t') \models \phi_2 \wedge \forall t'' \in (t,t'), (\rho,t'') \models \phi_1 \\
(\rho,t) \models \phi_1 \mathcal{S} \phi_2 & & \text{iff } \exists t' < t. (\rho,t') \models \phi_2 \wedge \forall t'' \in (t',t), (\rho,t'') \models \phi_1
\end{array}$$

The satisfiability problem for $TLCI_0$ is PSPACE-complete when the indices k of $C_k^{(0,b)}$ is coded in unary, and EXPSPACE-complete when the indices are coded in binary [18]. TLC is the special case where the upper bound b is 1.

Real-time logics are usually required to be scalable, in the sense that a change of the time scale (e.g. from second to minutes), or said otherwise the multiplication by a rational number, should not affect their definition. The logics presented here are not scalable, but their scalable version can be obtained by replacing the integers by rationals in the definition. This may change the expressiveness results below.

TLC was shown to be strictly more expressive than MITL with past with a simple non-scalable example [14]: It uses a single proposition p, that is true exactly at multiples of $2/3$. Every *MITL* formula with past will eventually behave like $p, \neg p, true$, or $false$. In contrast, $C_2^{(0,1)}(p)$ will be true on $\{((1+2i)/3, (2+2i)/3) | i \in \mathbb{N}\}$ indefinitely. Similarly, we see that the event clock y_p for p will always be between 0 and 1, but the memory event clock y_p^2 will periodically go above 1. In general, memory clocks bring the same supplementary expressive power, as we will see in the next section.

From [8] we can show that future TLC can be translated to the scalable MTL. To the best of our knowledge, the question whether TLC is more expressive than scalable MITL with past is open.

3.2 Recursive Memory Event Clocks Temporal Logic

In this section we introduce the Recursive Memory Event Clocks Temporal Logic (RMECTL). RMECTL extends EventClockTL of [20, 11]. We generalise its modalities by adding an index k: the recording modality $\lhd_I^k \phi$ means that the kth last time that

ϕ was true is in interval $t - I$, and symmetrically the predicting modality $\rhd_I^k \phi$ says the kth next occurrence of ϕ will occur within I. We count only one occurrence for an interval where ϕ is continuously true. Such a modality in fact introduces a memory event clock: $\lhd_I^k \phi$ means that we reset a memory clock each time ϕ is true, and we constrain the kth clock value at the time of evaluation. We denote the temporal logic where $k \leq n$ by RMECTL_n, for $n \in \mathbb{N}$. If we allow only index 1, we find back EventClockTL.

Definition 1. *The formulas of Recursive Memory Event Clock Temporal Logic (RMECTL) are built from propositional symbols $\mathbb{P}$, boolean connectives, the temporal operators* until *and* since *and two symmetric real-time modalities, the recording modality and predicting modality. The formulas ϕ of RMECTL are defined by the grammar:*

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

where p is a propositional symbol, $I \in \mathcal{I}_\mathbb{N}$ is an interval, and $n \in \mathbb{N}^+$. Let ϕ be a RMECTL formula and let ρ be a signal whose propositional symbols contain all propositions that occur in ϕ. The semantics of the new modalities are:

$(\rho, t) \models \lhd_I^n \phi$ *iff the set* $\{t_n \mid \exists t_1, \ldots, t_{n-1}, s_1, \ldots, s_{n-1} : t_n < s_{n-1} < t_{n-1} < \cdots < t_1 < t, \bigwedge_{i \leq n} (\rho, t_i) \models \phi, \bigwedge_{i<n} (\rho, s_i) \not\models \phi\}$ *Ends During* $t - I$

$(\rho, t) \models \rhd_I^n \phi$ *iff the set* $\{t_n \mid \exists t_1, \ldots, t_{n-1}, s_1, \ldots, s_{n-1} : t_n > s_{n-1} > t_{n-1} > \cdots > t_1 > t, \bigwedge_{i \leq n} (\rho, t_i) \models \phi, \bigwedge_{i<n} (\rho, s_i) \not\models \phi\}$ *Begins During* $t + I$

where "Begins During" and "Ends During" have been defined in Section 2.1. The intuition is that each t_i is a witness of an interval where ϕ was true, that caused a reset of the clock. They must be distinct intervals, i.e. they must be separated by an interval where ϕ is false, as witnessed by s_i. Intuitively, the nth previous reset is the maximum of the candidates t_n, but this maximum might not exist. Hence the indirect definition using "Begins During".

RMECTL turned out to be very close to TLCI_0:

Theorem 1. *RMECTL and $TLCI_0$ are intertranslatable linearly.*

Proof. **From RMECTL to TLCI_0** We first simplify formulas of RMECTL. First note that the left bound can always be set to 0 as follows: Define the downward closure of an interval I as $\downarrow I = \{t > 0 \mid \exists t' \in I . t \leq t'\}$. We use $\rhd_I^n \phi \equiv \neg \rhd^n_{\downarrow I \setminus I} \phi \wedge \rhd^n_{\downarrow I} \phi$. For instance, $\rhd^n_{(a,b]} \phi \equiv \neg \rhd^n_{(0,a]} \phi \wedge \rhd^n_{(0,b]} \phi$. Second, if the right bound of the interval is closed, we can open it. Define $\mathcal{J}\phi$ as $\phi \mathcal{U}\ true$. Intuitively, it means that ϕ will be true for some time just after the current point of time. Its dual $\mathcal{K}^+$ [10], i.e. $\neg(\neg\phi \mathcal{U}\ true)$, means that ϕ will be true *arbitrarily close* after the current point of time. Symmetrically, $\mathcal{B}\phi$, defined as $\phi \mathcal{S}\ true$, means that ϕ was true for some time just before now, and $\mathcal{K}^-$ [10], that ϕ will be true arbitrarily close before now. Then we use $\rhd^n_{(0,b]} \phi = \mathcal{J} \rhd^n_{(0,b)} \phi$ if ϕ is left-closed, which is expressed by $\neg\phi \mathcal{U} \phi$. This gives $\rhd^n_{(0,b]} \phi = (\neg\phi \mathcal{U} \phi \wedge \mathcal{J} \rhd^n_{(0,b)} \phi) \vee (\neg(\neg\phi \mathcal{U} \phi) \wedge \rhd^n_{(0,b)} \phi)$. We are only left with operators of the form $\rhd^n_{(0,b)} \phi$, which mean that n resets occur within interval $(0, b)$. A reset for a predicting clock is a rising edge, i.e. ϕ becomes true, and can be described by the formula: $(\mathcal{K}^- \neg\phi \wedge \phi) \vee (\neg\phi \wedge \mathcal{K}^+ \phi)$, that we abbreviate $\mathcal{R}\phi$. A

special case is when ϕ is true just after now ($\mathcal{K}^+\phi$), then $\rhd^1_{(0,b]}$ is true, even without a rising edge.Thus we translate $\rhd^n_{(0,b)}\phi$ by

$$(\neg\mathcal{K}^+\phi' \wedge C^{(0,b)}_n(\mathcal{R}\phi')) \vee (\mathcal{K}^+\phi' \wedge C^{(0,b)}_{n-1}(\mathcal{R}\phi'))$$

which is a formula of TLCI$_0$. $\lhd^n_{(0,b)}\phi$ is translated symmetrically. All other operators appear in both logics, and are translated trivially. This translation is linear in the number of subformulas, i.e. in DAG size which is the relevant measure for logics. It preserves or decreases the indices.

From TLCI$_0$ to RMECTL Let I be $(0, b)$. $C^{(0,b)}_n(\phi)$ means that there are at least n points satisfying ϕ in $t+(0, b)$, while $\rhd^n_{(0,b)}\phi$ means that $t+(0, b)$ comprises at least n rising edges (or $n - 1$ if it begins with a ϕ). This is different as soon as ϕ is true on a non-singular ϕ-interval. But then, this interval comprises an infinite number of ϕ points, and thus makes $C^{(0,b)}_n(\phi)$ true. Otherwise, all ϕ-intervals are singular, erasing the difference. Thus we translate $C^{(0,b)}_n(\phi)$ by $(\rhd_{(0,b)}\mathcal{J}\phi') \vee (\rhd^n_{(0,b)}\phi')$. $\overleftarrow{C}^{(0,b)}_n(\phi)$ is translated symmetrically. All other operators appear in both logics, and are translated trivially. This translation is linear in the number of subformulas (DAG size).

Corollary 1. *RMECTL and TLCI$_0$ have the same expressiveness.*

Corollary 2. *RMECTL is more expressive than MITL and EventClockTL.*

Note that MITL, TLC$_1$, and EventClockTL have the same expressiveness in interval or signal semantics [11].

Corollary 3. *Satisfiability and validity of RMECTL is PSPACE-complete if indices are in unary, EXPSPACE-complete if indices are in binary.*

3.3 The Temporal Logic of One Clock with Right Constraint

In this section, we introduce a fragment of the Timed Propositional Temporal Logic (TPTL) [5] that is expressively equivalent to TLC, but offers a more convenient syntax. TPTL is a temporal logic based on clock variables declared by "freeze quantifiers"; these clock variables can then be used in explicit real-time constraints. It is very natural and expressive, in particular more than MTL; hence it was dubbed "a really temporal logic" by its authors. Alas, satisfiability of full TPTL in most semantics is Σ^1_1-complete, i.e. highly undecidable [5]. The example TPTL formula below, borrowed from [4]:

$$Gx.(p \rightarrow F(q \wedge F(r \wedge x < 1)))$$

where $F\phi$ is $true\,\mathcal{U}\phi$, and G its dual, expresses quite naturally that each time we have p (a request), we will have q then r (it will be processed) within 1 second. This formula spurred further research [12, 18, 8] to express it in more decidable logics. Here, we propose instead a decidable fragment of TPTL that contains this example. Recently, another fragment was shown decidable, but only for the point semantics [17].

We use an adapted version of TPTL, called Clock Temporal Logic [19, p.46]: We interpret it with a continuous semantics on the non-negative reals (rather than a point semantics on the natural numbers [5]); we add the past modalities; we interpret the quantifiers as clock resets (rather than as a freezed copy of absolute time [5]).

Our fragment is called the One Clock Temporal Logic with constraints on the Right, abbreviated TPTL1R to evoke its link with TPTL. In this logic, only one clock can be active at a time. Furthermore, inside the scope of a clock, the formulae must be positive and until/since can only contain a constraint in the right side. The syntax is rather natural:

$$\phi ::= p \mid x.\phi_x \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{S} \phi_2$$
$$\phi_x ::= \phi \mid x \in I \mid \phi_x \vee \phi_x \mid \phi \wedge \phi_x \mid \phi_1 \mathcal{U} \phi_x \mid \phi_1 \mathcal{S} \phi_x$$

where ϕ_x means "a formula with clock variable x free", $p \in AP, I$ is an interval with integer bounds.

The semantics, $(\rho, t) \models \phi$, is defined as usual, plus for the reset quantifier:

$$(\rho, t) \models x.\phi_x \text{ iff } (\rho, t) \models_t \phi_x$$

For formulae ϕ_x, the real value v below is the time of the reset.

$$\begin{array}{lll}
(\rho, t) \models_v & \phi & \text{iff } (\rho, t) \models \phi \\
(\rho, t) \models_v & x \in I & \text{iff } t - v \in I \\
(\rho, t) \models_v & \phi'_x \vee \phi_x & \text{iff } (\rho, t) \models_v \phi'_x \text{ or } (\rho, t) \models_v \phi_x \\
(\rho, t) \models_v & \phi' \wedge \phi_x & \text{iff } (\rho, t) \models \phi' \text{ and } (\rho, t) \models_v \phi_x \\
(\rho, t) \models_v & \phi_1 \mathcal{U} \phi_x & \text{iff } \exists t' > t. (\rho, t') \models_v \phi_x \wedge \forall t'' \in (t', t). (\rho, t'') \models \phi_1 \\
(\rho, t) \models_v & \phi_1 \mathcal{S} \phi_x & \text{iff } \exists t' < t. (\rho, t') \models_v \phi_x \wedge \forall t'' \in (t', t), (\rho, t'') \models \phi_1
\end{array}$$

In the proof below, we use the fact that Q2MLO (called L_2 in [12]) and $TLCI_0$ have the same expressiveness [13]. Let us recall that Q2MLO is a first-order monadic logic of order. It contains first-order logic, plus a metric quantifier:

$$\phi ::= \forall t.\ \phi_1 \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid t_1 = t_2 \mid t_1 < t_2 \mid \exists t \in t_0 + I.\ \phi(t, t_0)$$

where I is a non-singular interval with integer bounds. Only two free variables t, t_0 are allowed in the quantified formula $\phi(t, t_0)$.

Theorem 2. *TPTL1R is as expressive as TLC, $TLCI_0$, and Q2MLO.*

Proof. It suffices to translate $TLCI_0$ to TPTL1R, and then TPTL1R to Q2MLO.

1. TPTL1R can express $C_n^{(0,b)}$:

$$C_n^{(0,b)}(\phi) = x.(F(\phi \wedge F(\phi \wedge \ldots F(\phi \wedge x < b))))$$

And symmetrically for the past operator.

2. The semantics of TPTL1R translates any ϕ_x appearing in $x.\phi_x$ into a first-order formula $\phi_x(v)$. We note that: (i) disjunctions are in the scope of existential quantifiers only, so that we can move the disjunctions out; (ii) each constraint $t_k - v \in I$ is also in the scope of existential quantifiers only, and in particular $\exists t_k$. Therefore the order of these existential quantifications is irrelevant, and we move in front $\exists t_k$ (the quantification whose variable appears in the constraint) then the other variables. The constraints are now part of a conjunction. We move each constraint just after its quantification. We obtain a formula of the form $\exists t_k.t_k - v \in I \wedge \phi(v, t_k)$, that we can express in Q2MLO as $\exists t_k \in v + I.\phi(v, t_k)$.

4 Recursive Memory Event Clocks Automata

As explained in the introduction, our goal is to extend in the realm of real-time the success of classical automata, that can express both specifications and programs uniformly, and are thus the internal data structure used by most model-checkers. Automata can deal with real-time by adding clocks that can be reset or tested. Timed automata allow liberal use of their clocks, making their inclusion problem undecidable. A more disciplined use of clocks is needed. Our proposal follows the idea of ECA. Since ECA reset clocks only on the occurrence of their atomic proposition, which is not very expressive, we proposed Recursive Event Clock Automata (RECA) [11]. "Recursive" refers to the fact that the resets of each clock of an automaton are controlled by a lower-level automaton. When this automaton visits a monitored location, it resets the associated clocks: An event-recording clock x_A and an event-predicting clock y_A can be associated with each monitored automaton. Thus no automaton can reset its own clocks. In particular, an automaton of level 0 has no subautomata, hence no clock.

Here, we examine whether introducing Memory Event Clocks (MEC) will further increase their power. This leads to Recursive Memory Event Clock Automata (RMECA). Event-recording memory clocks $x^i_{\mathcal{A}}$ record the time that has expired since the i_{th} last time at which the automaton $\mathcal{A}$ could pass through a monitored location, and the event-predicting memory clock $y^i_{\mathcal{A}}$ always records the amount of time that will expire until the i_{th} next time at which the automaton $\mathcal{A}$ could pass through a monitored location. Equivalently, a reset does not destroy the previous values of a memory clock: instead, a new clock with value 0 is created, and earlier clocks are still accessible by the indexed notation. We have already used MEC to define RMECTL in Section 3.2.

MEC are determined by the ISS (and not by the run as for TA). To deal with MEC, it is easier to consider them as supplementary propositions. We have then to make them mutually exclusive, so we consider atomic constraints. For a given clock c, let $R_c = \{r_1, \ldots, r_n\}$ be the constants to which it is compared in $\mathcal{A}$, in increasing order. The atomic constraints for c are $c = \bot, c < r_1, c = r_i, c \in (r_i, r_{i+1}), c > r_n$. An atomic constraint for $\mathcal{A}$ is a conjunction of atomic constraints for each clock. No clock valuation can satisfy two different atomic constraints, and each constraint of $\mathcal{A}$ can be expressed as a disjunction of atomic constraints. This latter property allows us to use only atomic constraints, using more locations if needed.

We now examine the complexities due to continuous time. We want these automata to consider ISS that define the same signal as equivalent, even if the intervals might be

split differently. This goal will force the definition of deterministic automata below. To simplify it, we label our locations not only with atomic propositions but also with "limits". Intuitively, a past (resp. future) limit describes what happened just before (resp. just after) the current time. Locations where some limit is different from the current label are called *singular*: only a single instant can be spent there. From a singular location, we can only make a transition to a non-singular location, where the labelling must be as predicted by the limit of the singular location. The past limit of $\top$ is false only in initial locations.

Given a set propositions $\mathbb{P}$, the limit closure (*Limit*($\mathbb{P}$)) is the set $\{p, \overrightarrow{p}, \overleftarrow{p} \mid p \in \mathbb{P} \cup \{\top\}\}$. $\overrightarrow{p}$ is the future limit of p and $\overleftarrow{p}$ is the past limit of p.

Definition 2. *A Recursive Memory Event Clock Automaton (RMECA) is a tuple $\mathcal{A} = (\mathbb{P}, S, S_0, \rightarrow, C, \gamma, M, F)$, such that:*

1. *$\mathbb{P}$ is a set finite set of propositional symbols.*
2. *S is a finite set of locations and $S_0 \subseteq S$ is the set of starting locations.*
3. *$\rightarrow \subseteq S \times S$ are the transitions.*
4. *A finite set of atomic constraints C, containing clocks x_B^i or y_B^i, with B a lower-level RMECA.*
5. *$\gamma : S \rightarrow 2^{Limit(\mathbb{P} \cup C)}$ is a function which labels each location $s \in S$ with the set of limits of propositions and constraints that are true in that location.*
6. *$M \subseteq S$ is the set of monitored locations: when the automaton visits such a location, it resets the associated clock.*
7. *$F \subseteq S$ is a set of Büchi accepting locations.*

We now define when a RMECA accepts an ISS ρ, thanks to a timed run. This is the time t when the automaton can visit a monitored location.

Definition 3. *A RMECA $\mathcal{A}$ accepts a signal ρ at time t, if there exist an infinite timed run $\theta = (s, I)$ such that following conditions holds:*

1. *the run starts in a starting location $s_0 \in S_0$.*
2. *for all $i > 0$, the run either follows a transition: $s_{i-1} \rightarrow s_i$ or stutters: $s_{i-1} = s_i$.*
3. *It is in a monitored location at time t: $\theta(t) \in M$.*
4. *The labelling of the location corresponds to the ISS and satisfies the limits and clock constraints: $\forall t' \in \mathbb{R}^+, (\rho, t) \models \gamma(\theta(t))$.*
5. *It visits infinitely often a Büchi location.*

The clock valuation function over a lower-level RMECA at $\mathcal{A}$ and time t at ρ, is noted $\nu_t^\rho : \mathcal{C}_\mathcal{A} \rightarrow \mathbb{R}_\perp^+$. It assigns a (non-standard) positive real, or undefined, to each clock variable. The resets are done when $\mathcal{A}$ can visit a monitored location. Given t and ρ, the reset interval of $y_\mathcal{A}^n$ is the interval I_n such that there are non-empty intervals $I_1, \cdots, I_{n-1}$ where:

1. $t < I_1 < \cdots < I_n$,
2. $\forall i \leq n, \forall r \in I_i, \mathcal{A}$ accepts ρ in r
3. $\forall i < n, BetwI(I_i, I_{i+1})$ is not empty and $\forall r \in BetwI(I_i, I_{i+1}), \mathcal{A}$ does not accept ρ in r

4. $\forall r \in BetwI(\{t\}, I_1), \mathcal{A}$ does not accept ρ in r.

Note that I_1 might begin just after t, in which case the last condition is vacuously true. The reset interval for a recording clock is symmetric. The value of a clock is:

$$\nu_t^\rho(x_\mathcal{A}^n) = \begin{cases} t - r & \text{if the reset interval of } x_\mathcal{A}^n \text{ is } (l, r] \text{ or } [l, r] \\ (t - r)^+ & \text{if the reset interval of } x_\mathcal{A}^n \text{ is } (l, r) \text{ or } [l, r) \\ \perp & \text{if } x_\mathcal{A}^n \text{ has no reset interval} \end{cases}$$

Symmetrically,

$$\nu_t^\rho(y_\mathcal{A}^n) = \begin{cases} l - t & \text{if the reset interval of } y_\mathcal{A}^n \text{ is } [l, r) \text{ or } [l, r] \\ (l - t)^+ & \text{if the reset interval of } y_\mathcal{A}^n \text{ is } (l, r) \text{ or } (l, r] \\ \perp & \text{if } y_\mathcal{A}^n \text{ has no reset interval} \end{cases}$$

The logic RMECTL and RMECA in fact use the same memory clocks:

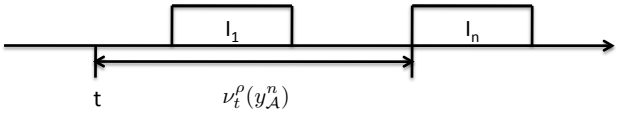

Fig. 1. The value of $y_\mathcal{A}^n$ and its reset interval I_n.

Theorem 3. $\nu_t^\rho(x_\mathcal{A}^n) \in I$ *iff* $(\rho, t) \models \lhd_I^n p$
where p *is a proposition such that* $(\rho, t) \models p$ *iff* $\mathcal{A}$ *accepts* ρ *at time* t.

4.1 Properties of *RMECA*

We now show here that RMECA inherit all good properties of RECA and ECA: they are determinizable and closed over all boolean operations. The proofs are the same as for RECA [19], replacing clocks by memory clocks. In view of the fact that we later show that they are expressively equivalent, this seems obvious, but (i) we need those properties to prove the equivalence, and (ii) the direct proofs give algorithms that are more efficient, since the translation of the next section is exponential.

We first adapt the definition of determinism to cater for continuous time [19]:

Definition 4. *A RMECA* $\mathcal{A} = (\mathbb{P}, S, S_0, \to, C, \gamma, M, F)$*, is deterministic iff:*

1. *Distinct initial locations* $s_1 \neq s_2 \in S_0$ *have distinct labellings:* $\gamma(s_1) \neq \gamma(s_2)$
2. *Successive locations* $s_1 \to s_2$ *have distinct labellings.*
3. *Distinct successor locations* $s_2 \neq s_3, s_1 \to s_2, s_1 \to s_3$ *have distinct labellings.*

The determinism ensures that, at each time t during a run, the choice of the next location is uniquely determined by the current location of the automaton and (ρ, t). Condition (2) ensures that the time at which to leave a location is uniquely given by the signal ρ. Therefore there is at most one (signal) run for each ρ.

Theorem 4. *For any RMECA $\mathcal{A}$, we construct a deterministic Rabin RMECA $\mathcal{A}'$ that accepts the same language. If $\mathcal{A}$ has n locations, $\mathcal{A}'$ has $2^{O(nlogn)}$ locations and the same clocks and subautomata.*

Note that the time of acceptance also preserved by determinization.

Theorem 5. *The class of RMECA-recognizable timed languages is closed under union, intersection and complementation.*

This differs from RECA and ECA, where those problems are PSPACE-complete. The higher complexity is only due to the indices n of the clocks, that can be expressed compactly (in $\log n$) in binary notation, while their implementation requires n clocks. If the indices are expressed in unary notation, these problems are PSPACE-complete.

Theorem 6. *The inclusion problem for RMECA are EXPSPACE-Complete.*

Proof. Consider two RMECA $\mathcal{A}$ and $\mathcal{B}$ such that each automaton has at most n locations, let m be the number of the clocks, let k be the highest index of a memory clocks. Let c be the largest integer constant that appears in the clock constraints. To check whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$, we first to determinize $\mathcal{B}$ to $\mathcal{B}_1$, after we complement $\mathcal{B}_1$ to $\mathcal{B}_2$. The automata $\mathcal{B}_2$ has $2^{(n\cdot k)\cdot\log(n\cdot k)}$ location since it is a Büchi automaton and the integer constants that appear in the clock constraints of $\mathcal{B}_2$ are bounded by c. Let $\mathcal{D}$ be the product of $\mathcal{A}$ and $\mathcal{B}_2$. The *RMECA* $\mathcal{D}$ has $n \cdot 2^{(n\cdot k)\cdot\log(n\cdot k)}$ locations, where the integer constants that appear in the clock constraints of $\mathcal{D}$ are also bounded by c. Now, we can use the MRECA $\mathcal{D}$, and check for its emptiness. Since checking emptiness of RMECA is exponential-sized finite state machine, it follows that emptiness of $\mathcal{D}$ can be checked in EXPSPACE. The proof of hardness is the same as the proof for RECA [19].

Theorem 7. *The emptiness, universality, and language inclusion problems for RMECA are EXPSPACE-Complete.*

4.2 From RMECA to RECA

Memory clocks allow to measure distances from several resets. But do they really increase the power of when placed inside automata? The answer is positive for ECA, but negative for RECA and TA. Intuitively, automata can already count resets (though less concisely). For an example, consider again the formula [4]:

$$Gx.(p \rightarrow F(q \wedge F(r \wedge x < 1)))$$

Assuming that p and q do not occur together on a non-singular interval (which is always the case in point semantics), this formula can be translated to the RECA of Fig. 2, that counts modulo 2. To save space, we have drawn the main automaton and its two subautomata $A1, A2$ with the same transitions. All states are accepting. $A1$ is a copy of the figure where clocks constraints are removed, and the location marked M_{A1} is the only monitored location, and similarly for $A2$. The main automaton has the clock constraints, but no monitored locations. Although the formula conceptually starts an infinite number of clocks x, this automaton shows that two clocks suffice.

More generally, we can use counting to eliminate memory event clocks:

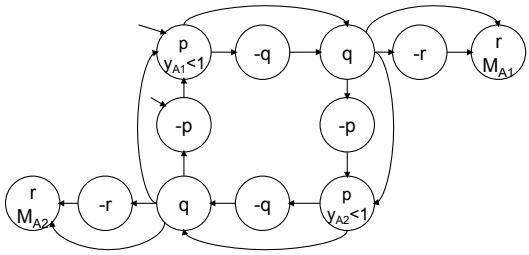

Fig. 2. A RECA for $Gx.(p \to F(q \wedge F(r \wedge x < 1)))$

Theorem 8. *For any RMECA, we can construct a RECA that accepts the same language.*

Proof. Let $\mathcal{A}$ be the initial RMECA. To simulate a memory clock $x^n_{\mathcal{C}}$ where $\mathcal{C}$ is a subautomaton of $\mathcal{A}$, we must keep track of the last n events and thus use n clocks. The clocks must be used in a rotating manner, since we cannot copy them. We can assume that $\mathcal{C}$ is deterministic, since RMECA are determinizable. To reset them, we augment $\mathcal{C}$ to count modulo n, and make n copies $\mathcal{C}'(i)$ with monitored states that differ by the count only.

Let $\mathcal{C}'(i)$ (with $i < n$) be the automaton:

1. Its symbols are the same: $\mathbb{P}' = \mathbb{P}$
2. Its states $S' = S \times [0..n)$ are pairs (s, j) where s is a state of $\mathcal{C}$, and $j < n$ counts the number of times we entered a monitored region, modulo n;
3. $S'_0 = S_0 \times \{0\}$;
4. The copy $\mathcal{C}'(i)$ monitors states of count i: $S'_m = S_m \times \{i\}$;
5. The transitions update the count when entering a monitored region: $(s, j) \to (s', j')$ iff $s \to s', j = j', (s \in S_m$ if $s' \in S_m)$ or $s \to s', j' = j + 1 \bmod n, s \notin S_m, s' \in S_m$;
6. The labellings are unchanged: $\gamma'((s, j)) = \gamma(s)$;
7. The accepting states are unchanged: $F' = F \times \{0..n-1\}$.

Each $\mathcal{C}'(i)$ is bisimilar with $\mathcal{C}$. $\mathcal{C}$ visits a monitored state iff (exactly) one of the $\mathcal{C}'(i)$ does. Therefore the set of values of the clocks is the same:

$$\{\nu^\rho_t(x_{\mathcal{C}'(i)}) \mid i < n\} = \{\nu^\rho_t(x^j_{\mathcal{C}}) \mid 0 < j \leq n\}$$

Now we translate the constraints that appear in the upper level automaton: the value of $x^n_{\mathcal{C}}$ is the maximal value of this set. Thus a constraint $x^n_{\mathcal{C}} < c$ is translated by $\bigwedge_{i<n} x_{\mathcal{C}'(i)} < c$ (idem for $\leq$), symmetrically $x^n_{\mathcal{C}} > c$ by $\bigvee_{i<n} x_{\mathcal{C}'(i)} > c$ (idem for $\geq$), and $x^n_{\mathcal{C}} = \bot$ by $\bigvee_{i<n} x_{\mathcal{C}'(i)} = \bot$. $x = c$ is considered as an abbreviation for $x \geq c \wedge x \leq c$.

4.3 From RMECTL to RMECA

We briefly present the construction of a RMECA from a RMECTL formula.

Theorem 9. *For every RMECTL formula ϕ, we can construct a RMECA $\mathcal{A}_\phi$ that accepts ρ at time t iff $(\rho, t) \models \phi$.*

The construction is as in [19], with memory clocks instead of clocks. The transformation is done level by level, where the level of a formula is the nesting depth of real-time modalities. A formula $\rhd_I^n \phi$ is translated as constraint $x^n_{\mathcal{A}_\phi} \in I$. The formula ϕ is recursively transformed in a tableau automaton for continuous time, where the monitored states are the states containing ϕ. The construction is exponential in the size of the non-real time part of the formula, but linear in the real-time part.

4.4 Region construction

The principles of the called region automaton which transforms a timed automaton $\mathcal{A}$ ($\mathcal{R}(\mathcal{A})$) into an untimed finite automaton can be applied to RMECA[3][19]. The following theorem is the basis for an algorithmic analysis de RMECA automata:

Construction 1 Let $\mathcal{M}$ be a RECA and let m the number of clocks, n is the number of locations. Let c be the largest integer constant that appears in the clock constraints. From [19], it follows that the number of locations in the region automaton is $n \cdot 2^n \cdot 2^{O(m \cdot \log(m \cdot c))}$. The inclusion problem for RECA is in fact PSPACE-complete. In case of RMECA, we will use *theorem 7*, we first construct a RECA that accepts the same language of RMECA. In this case, as the number of clocks are increased and hence the number of clock regions increases to $n \cdot 2^n \cdot 2^{O((m \cdot k) \cdot \log((m \cdot k) \cdot c))}$, where k is the highest index of a memory clocks.

Theorem 10. *The number of location in the region automaton $\mathcal{R}(\mathcal{A})$ of a RMECA automata $\mathcal{A}$ is finite $n \cdot 2^{O((m \cdot k) \cdot \log((m \cdot k) \cdot c))}$, where n is the number of locations in $\mathcal{A}$, m is the number of clocks, k is the highest index of a memory clock and c is the largest constant appearing in $\mathcal{A}$.*

Theorem 11. *The language of $\mathcal{R}(\mathcal{A})$ corresponds to Untimed($\mathcal{L}(\mathcal{A})$). Consequently, the timed language of $\mathcal{A}$ is empty iff the language of $\mathcal{R}(\mathcal{A})$ is empty.*

5 Conclusions

Recursive event clocks have been criticized to be too weak as they only record the time to next/previous event. In this paper, we have introduced memory event clocks that are designed to overcome this limitation. We presented them in the interval-based semantics, both to ease comparison with related work and since this setting is more general and more difficult than point semantics.

When we introduce such clocks in a temporal logic, we obtain RMECTL. RMECTL allows punctuality constraints and allows constraints on the n_{th} next (n_{th} last) time a

formula will be (was) true. Still, we have shown that RMECTL has the same expressiveness and complexity as TLCI_0.

We also identified a fragment of TPTL that is expressively equivalent and decidable. We conjecture that a larger fragment of TPTL is decidable.

The operational nature of these clocks blends nicely with automata, giving RMECA. They keep all the nice properties of the original event clock automata. They are as expressive as our RECA [11], showing that TLC was already included in RECA, under finite variability. The increase of expressive power is thus modest enough to disappear at automata level. In other words, the criticism was not founded with respect to RECA.

Automata are known to be equivalent to second-order quantification, and this opens the corresponding logical question, whether Q-MITL and Q-TLC, (i.e. with second-order quantification that does not to cross scope with real-time operators) are equivalent. Our results settles this question only under finite variability. Another open question is to characterize the strongest first-order real-time temporal logic included in RECA, beyond TLC perhaps. It is also open, whether TLC is more expressive than scalable MITL with past.

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, and by the European Science Foundation (ESF) under EUROCORES project LogiCCC/GASICS.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
2. R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *ACM*, 43(1):116–146, 1996.
3. R. Alur, L. Fix, and T. A. Henzinger. A determinizable class of timed automata. In *CAV*, volume 818 of *LNCS*, pages 1–13. Springer, 1994.
4. R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *REX Workshop*, volume 600 of *LNCS*, pages 74–106. Springer, 1991.
5. R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1994.
6. A. Annichini, A. Bouajjani, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In *Proc. 13th Int. Conference on Computer Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 368–372. Springer, 2001.
7. C. Baier, N. Bertrand, P. Bouyer, and T. Brihaye. When are timed automata determinizable? In *ICALP (2)*, volume 5556 of *LNCS*, pages 43–54. Springer, 2009.
8. P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of tptl and mtl. In R. Ramanujam and S. Sen, editors, *FSTTCS*, volume 3821 of *LNCS*, pages 432–443. Springer, 2005.
9. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Proc. 10th Int. Conference on Computer Aided Verification (CAV)*, volume 1427 of *LNCS*, pages 546–550. Springer, 1998.
10. D. M. Gabbay and I. M. Hodkinson. An axiomatization of the temporal logic with until and since over the real numbers. *J. Log. Comput.*, 1(2):229–259, 1990.

11. T. A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In *ICALP*, volume 1443 of *LNCS*, pages 580–591. Springer, 1998.
12. Y. Hirshfeld and A. M. Rabinovich. A framework for decidable metrical logics. In *ICALP*, volume 1644 of *LNCS*, pages 422–432. Springer, 1999.
13. Y. Hirshfeld and A. M. Rabinovich. An expressive temporal logic for real time. In *MFCS*, volume 4162 of *LNCS*, pages 492–504. Springer, 2006.
14. Y. Hirshfeld and A. M. Rabinovich. Expressiveness of metric modalities for continuous time. In *CSR*, volume 3967 of *LNCS*, pages 211–220. Springer, 2006.
15. J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *LICS*, pages 54–63. IEEE, 2004.
16. J. Ouaknine and J. Worrell. Some recent results in metric temporal logic. In *FORMATS*, volume 5215 of *LNCS*, pages 1–13. Springer, 2008.
17. P. Parys and I. Walukiewicz. Weak alternating timed automata. In *ICALP (2)*, volume 5556 of *LNCS*, pages 273–284. Springer, 2009.
18. A. Rabinovich. Complexity of metric temporal logics with counting and the Pnueli modalities. In *FORMATS*, volume 5215 of *LNCS*, pages 93–108. Springer, 2008.
19. J.-F. Raskin. *Logics, Automata and Classical Theories for Deciding Real Time*. Phd thesis, FUNDP University, Belgium, 1999.
20. J.-F. Raskin and P.-Y. Schobbens. State clock logic: A decidable real-time logic. In *HART*, volume 1201 of *LNCS*, pages 33–47. Springer, 1997.
21. The UPPAAL tool. Available at `http://www.uppaal.com/`.