Model-Checking Timed Temporal Logics

Patricia Bouyer
LSV, CNRS & ENS Cachan, France
bouyer@lsv.ens-cachan.fr

Abstract

In this paper, we present several timed extensions of temporal logics, that can be used for model-checking real-time systems. We give different formalisms and the corresponding decidability/complexity results. We also give intuition to explain these results.

1 Introduction

Timed automata [3, 4] are a well-established model for real-time systems. One of the most important properties they enjoy is probably that reachability properties can be decided in that class of systems. This has given rise to multiple works, both on theoretical aspects and on more practical and algorithmic aspects. Several tools have even been developed for model-checking timed automata, for instance HyTech [37], Kronos [30] or Uppaal [42, 14]. Moreover there are already several success stories, we only mention the correction and verification of the Bang & Olufsen audio/video protocol [35] made using the tool Uppaal.

Timed automata are adequate to represent systems, but not that much for representing properties of systems. If $\mathcal{A}$ is a timed automaton representing the system, and $\mathcal{P}$ a timed automaton representing the property, verifying that $\mathcal{A}$ satisfies the property $\mathcal{P}$ corresponds to checking that all behaviours of $\mathcal{A}$ are also behaviours of $\mathcal{P}$. This is an inclusion question, and that problem is unfortunately undecidable for timed automata [4].

Hence, following the development of temporal logics in model-checking [29, 51], timed temporal logics have been proposed, which extend classical untimed temporal logics with timing constraints. There are several ways of expressing such constraints, a standard one consists in constraining temporal modalities. For instance, we can write a formula like

$$\mathbf{G}\,(\mathsf{problem} \rightarrow \mathbf{F}_{\leq 5\,\mathrm{min}}\,\mathsf{repair})$$

to express the following *quantitative* property: every time there is a problem, it must be repaired within 5 minutes. This kind of properties cannot be expressed using standard temporal logics, as those logics can only speak about the relative order of events, not about the distance (in time) between these events. Several timed extensions of CTL [29] and LTL [51] have been proposed, and TCTL, a natural extension of CTL, has been first proved suitable (from a decidability point-of-view) for model-checking [1, 2]. On the contrary, until recently [48], timed extensions of LTL were somewhat improperly considered as too hard for being used for model-checking purposes [11, 36].

In this paper, we present some of the timed extensions of temporal logics that have been studied in the purpose of model-checking real-time systems. We first briefly present TCTL [1], a branching-time timed temporal logic, and then focus on linear-time timed temporal logics, like MTL [41] and variants or subclasses. For those logics, we give decidability and complexity results for the model-checking problem. We also give intuitive explanations to these results.

Those logics, presented here in the perspective of model-checking, are also pretty much studied from a more logic and automata theoretical point-of-view. Indeed, they can be viewed as logics over the reals, and it is natural to embed them in a more general context than model-checking, looking at questions like expressive completeness, axiomatization, satisfiability problems, equivalence with other formalisms, *etc*. We just point to few works, [52, 54, 39, 28], but the list of papers on that subject is indeed very long.

2 Preliminaries

The model of timed automata has been defined in the 90s by Alur and Dill as a model for representing systems with real-time constraints [4]. A timed automaton is basically a finite automaton which manipulates finitely many variables called clocks.

Let X be a finite set of *clocks*. A *clock valuation* over X is a mapping $v : X \rightarrow \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of nonnegative reals. We write $\mathbb{R}_+^X$ for the set of clock valuations over

X. If $v \in \mathbb{R}_+^X$ and $\tau \in \mathbb{R}_+$, $v+\tau$ is the clock valuation defined by $(v+\tau)(x) = v(x)+\tau$ if $x \in X$. If $Y \subseteq X$, the valuation $[Y \leftarrow 0]v$ is the valuation assigning 0 to $x \in Y$ and $v(x)$ to $x \notin Y$. A *guard* over X is a finite conjunction of expressions of the form $x \sim c$ where $x \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$. We denote by $\mathcal{G}(X)$ the set of guards over X. The satisfaction relation for guards over clock valuations is defined in a natural way, and we write $v \models g$, whenever v satisfies g. We denote AP a finite set of atomic propositions.

Definition 1 *A* timed automaton *is a tuple* $\mathcal{A} = (L, \ell_0, X, E, I, \mathcal{L})$ *such that:*

(i) *L is a finite set of locations,*

(ii) *$\ell_0 \in L$ is the initial location,*

(iii) *X is a finite set of clocks,*

(iv) *$E \subseteq L \times \mathcal{G}(X) \times 2^X \times L$ is a finite set of edges,*

(v) *$I : L \to \mathcal{G}(X)$ is an invariant,*

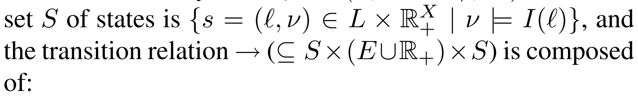
(vi) *$\mathcal{L} : L \to 2^{\mathsf{AP}}$ is a labelling function.*

The semantics of a timed automaton $\mathcal{A}$ is given as a labelled transition system $T_{\mathcal{A}} = (S, E \cup \mathbb{R}_+, \to)$ where the set S of states is $\{s = (\ell, \nu) \in L \times \mathbb{R}_+^X \mid \nu \models I(\ell)\}$, and the transition relation $\to (\subseteq S \times (E \cup \mathbb{R}_+) \times S)$ is composed of:

- *(delay transition)* $(\ell, \nu) \xrightarrow{\tau} (\ell, \nu+\tau)$ if $\tau \in \mathbb{R}_+$, and for all $0 \leq \tau' \leq \tau$, $\nu + \tau' \models I(\ell)$,
- *(discrete transition)* $(\ell, \nu) \xrightarrow{e} (\ell', \nu')$ if $e = (\ell, g, Y, \ell') \in E$ is such that $\nu \models I(\ell) \wedge g$, $\nu' = [Y \leftarrow 0]\nu$, and $\nu' \models I(\ell')$.

A *finite (resp. infinite) run* ϱ of $\mathcal{A}$ is a finite sequence of states obtained by alternating delay and discrete transitions, *i.e.*, $\varrho = s_0 \xrightarrow{\tau_1} s_1' \xrightarrow{e_1} s_1 \xrightarrow{\tau_2} s_2' \xrightarrow{e_2} s_2 \cdots s_{n-1} \xrightarrow{\tau_n} s_n' \xrightarrow{e_n} s_n \cdots$ or more compactly $s_0 \xrightarrow{\tau_1, e_1} s_1 \xrightarrow{\tau_2, e_2} s_2 \cdots s_{n-1} \xrightarrow{\tau_n, e_n} s_n \cdots$. In the following, we may want to interpret such a run as the *timed word* $(a_0, 0)(a_1, t_1)(a_2, t_2) \cdots (a_n, t_n) \cdots$ where for each $i \geq 0$, $a_i = \mathcal{L}(s_i)$ ($\mathcal{L}$ is straightforwardly extended from L to S), and $t_i = \sum_{j \leq i} \tau_j$.

Example 1 *Consider the timed automaton depicted below:*



A possible run of that automaton is:

$$(\ell_0, (0,0)) \xrightarrow{3.2} (\ell_0, (3.2, 3.2)) \to (\ell_1, (3.2, 0)) \xrightarrow{1.5} (\ell_1, (4.7, 1.5)) \to (\ell_2, (4.7, 1.5))$$

The notation $(\ell_1, (4.7, 1.5))$ *means that the automaton is in state* ℓ_1*, that the value of clock* x *is* 4.7 *and the value of clock* y *is* 1.5.

3 Branching-Time Timed Temporal Logics

3.1 Syntax and semantics

The branching-time logic TCTL,[1] which extends the classical untimed branching-time logic CTL [29] with time constraints on modalities, has been defined

The syntax of TCTL is given by the following grammar:

$$\mathsf{TCTL} \ni \varphi ::= a \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{E}\,\varphi\,\mathbf{U}_I\,\varphi \mid \mathbf{A}\,\varphi\,\mathbf{U}_I\,\varphi$$

where $a \in \mathsf{AP}$, and I is an interval of $\mathbb{R}_+$ with integral bounds.

There are two possible semantics for TCTL, one which is said *'continuous'*, and the other one which is more discrete and is said *'pointwise'*. These two semantics share rules for basic modalities, and only differ in the interpretation of the term *'position'*:

$$\begin{aligned}
(\ell, v) \models a &\Leftrightarrow a \in \mathcal{L}(\ell)\\
(\ell, v) \models \neg\varphi &\Leftrightarrow (\ell, v) \not\models \varphi\\
(\ell, v) \models \varphi \vee \psi &\Leftrightarrow (\ell, v) \models \varphi \text{ or } (\ell, v) \models \psi\\
(\ell, v) \models \mathbf{E}\,\varphi\,\mathbf{U}_I\,\psi &\Leftrightarrow \text{there is an infinite run } \varrho \text{ in } \mathcal{A}\\
&\qquad \text{from } (\ell, v) \text{ such that } \varrho \models \varphi\,\mathbf{U}_I\,\psi\\
(\ell, v) \models \mathbf{A}\,\varphi\,\mathbf{U}_I\,\psi &\Leftrightarrow \text{any infinite run } \varrho \text{ in } \mathcal{A} \text{ from } (\ell, v)\\
&\qquad \text{is such that } \varrho \models \varphi\,\mathbf{U}_I\,\psi\\
\varrho \models \varphi\,\mathbf{U}_I\,\psi &\Leftrightarrow \text{there exists a position } \pi > 0\\
&\qquad \text{along } \varrho \text{ such that } \varrho[\pi] \models \psi,\\
&\qquad \text{for every position } 0 < \pi' < \pi,\\
&\qquad \varrho[\pi'] \models \varphi, \text{ and } \mathsf{duration}(\varrho_{\leq\pi}) \in I
\end{aligned}$$

where $\varrho[\pi]$ is the state of ϱ at position π, $\varrho_{\leq\pi}$ is the finite prefix of ϱ ending at position π, and $\mathsf{duration}(\varrho_{\leq\pi})$ is the sum of all delays along ϱ up to position π. The $\mathbf{U}$-modality is called the 'Until' operator.

In the continuous semantics, a position in a run ϱ is any state appearing along ϱ.[2] For instance if there is a transition $(\ell, v) \xrightarrow{\tau, e} (\ell', v')$ in ϱ, then any state $(\ell, v+t)$ with $0 \leq t \leq \tau$ is a position of ϱ, and obviously, so is (ℓ', v'). This semantics is very strong because for ϱ to satisfy $\varphi\,\mathbf{U}_{\sim c}\,\psi$, all intermediary states of ϱ need to satisfy φ before ψ holds.

In the pointwise semantics, a position in a run $\varrho = s_0 \xrightarrow{\tau_1, e_1} s_1 \xrightarrow{\tau_2, e_2} s_2 \cdots s_{n-1} \xrightarrow{\tau_n, e_n} s_n \cdots$ is an integer i and the corresponding state s_i. In this semantics, formulas are checked only right after a discrete action has been done. Sometimes, the pointwise semantics is given in terms

[1] TCTL stands for "Timed Computation Tree Logic" and has been defined in [1].

[2] That can be formalized, see for instance [16].

of actions and timed words, but it does not change anything. Later, we may sometimes use the timed words terminology.

As usually in CTL, we define syntactic sugar to TCTL: $\mathtt{tt} \equiv a \vee \neg a$ standing for true, $\mathtt{ff} \equiv \neg\mathtt{tt}$ standing for false, the implication $\varphi \to \psi \equiv (\neg\varphi \vee \psi)$, the eventuality operator $\mathbf{F}_I\,\varphi \equiv \mathtt{tt}\,\mathbf{U}_I\,\varphi$, and the globally operator $\mathbf{G}_I\,\varphi \equiv \neg(\mathbf{F}_I\,\neg\varphi)$.

Example 2 *In* TCTL*, we can write many kinds of properties, for instance, bounded-response time properties like*

$$\mathbf{A}\,\mathbf{G}\,(a \to \mathbf{A}\,\mathbf{F}_{\leq 56}\,b) \qquad (1)$$

expressing that each time a holds, along all possible runs, b has to hold within 56 *time units.*

Remark 1 *In [38],* TCTL *is given with external clock variables. That is, we can use variables to express timing constraints. We will not give the precise grammar of that version of* TCTL*, but just give the equivalent of formula (1) in this framework:*

$$\mathbf{A}\,\mathbf{G}\,(a \to x.\mathbf{A}\,\mathbf{F}\,(b \wedge x \leq 56))$$

The interpretation of that formula is the following: each time an a is encountered, we reset a clock x, and check that along all possible runs, later, b holds and the value of the clock x (which has increased at the same speed as the universal time) is not more than 56.

In [19], it has been proved that TCTL *with external clock variables is strictly more expressive than* TCTL *with intervals constraining the modalities.*

3.2 Decidability

The decidability and complexity of the model-checking problem for TCTL is known since the beginning of the 90s:

Theorem 1 *The model-checking problem for* TCTL *(under the continuous or the pointwise semantics) is* PSPACE*-Complete [1, 2].*

This result relies on the construction of a region automaton construction, slight extension of the classical region automaton construction [3, 4] for deciding language emptiness in timed automata. The region automaton is a finite automaton which abstracts away the value of the clocks, and which recognizes the untimed projection of the timed language recognized by the original timed automaton[3] Then, the model-checking algorithm for TCTL consists in labelling the states of the region automaton[4] with formulas that are satisfied from the corresponding states in the original timed automaton. In particular, the proof settles that two states which are in the same region satisfy the same TCTL formulas.

We do not give more details for the model-checking of TCTL formulas, as these results are known since a long time, but better turn to the linear-time framework.

4 Linear-Time Timed Temporal Logics

Like classical temporal logics, linear-time property languages have also been studied in the framework of timed systems. Two main temporal formalisms have been defined: (i) MTL [5] is the counterpart of TCTL without external clock variables, and extends LTL [51] by adding timing constraints on modalities; (ii) TPTL [6] extends LTL by adding timing constraints to specifications using external variables and constraints thereon.

4.1 Syntax and semantics of MTL

The syntax of MTL is given by the following grammar:

$$\mathsf{MTL} \ni \varphi ::= a \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\,\mathbf{U}_I\,\varphi$$

where $a \in \mathsf{AP}$, and I is an interval of $\mathbb{R}_+$ with integral bounds.

It will even be more important in the context of linear-time timed temporal logics: as for TCTL, we distinguish between the two semantics, pointwise and continuous. Let $s_0 \xrightarrow{\tau_1,e_1} s_1 \xrightarrow{\tau_2,e_2} s_2 \cdots s_{n-1} \xrightarrow{\tau_n,e_n} s_n \cdots$ with $s_0 = (\ell_0, v_0)$ be a finite or infinite run. Then:

$$\begin{aligned}
\varrho \models a &\Leftrightarrow a \in \mathcal{L}(\ell_0)\\
\varrho \models \neg\varphi &\Leftrightarrow \varrho \not\models \varphi\\
\varrho \models \varphi \vee \psi &\Leftrightarrow \varrho \models \varphi \text{ or } \varrho \models \psi\\
\varrho \models \varphi\,\mathbf{U}_I\,\psi &\Leftrightarrow \text{there exists a position } \pi > 0 \text{ along } \varrho \text{ s.t.}\\
&\qquad \varrho[\pi] \models \psi,\ \text{for every position } 0 < \pi' < \pi,\\
&\qquad \varrho[\pi'] \models \varphi,\ \text{and } \mathsf{duration}(\varrho_{\leq\pi}) \in I
\end{aligned}$$

with the same distinctions for the term 'position', depending on the choice of the semantics.

As for LTL, we define some syntactic sugar for MTL: $\mathtt{tt} \equiv (a \vee \neg a)$ stands for true, $\mathtt{ff} \equiv (\neg\mathtt{tt})$ stands for false, $(\varphi \to \psi) \equiv (\neg\varphi \vee \psi)$, $\mathbf{F}_I\,\varphi \equiv (\mathtt{tt}\,\mathbf{U}_I\,\varphi)$ (eventually, φ will hold within interval I from now), $\mathbf{G}_I\,\varphi \equiv \neg(\mathbf{F}_I\,\neg\varphi)$ (for all positions within I, φ holds), and $\mathbf{X}_I\varphi \equiv (\mathtt{ff}\,\mathbf{U}_I\,\varphi)$ (next position is within I from now and satisfies φ). Moreover, we use pseudo-arithmetical espressions to represent intervals. For instance, '$= 1$' stands for the singleton interval $[1, 1]$, and '≥ 2' stands for the interval $[2, +\infty)$.

[3]The timed language recognized by a timed automaton $\mathcal{A}$ (with a distinguished acceptance condition, as in the context of finite automata) is the set of timed words that can be read in $\mathcal{A}$ on a run satisfying the acceptance condition.

[4]Formally, it is the region automaton extended with an extra clock which will measures delays that will be checked by the formula.

[5]MTL stands for "Metric Temporal Logic" and has been first proposed by Koymans [41].

[6]TPTL stands for "Timed Propositional Temporal Logic", and has been first proposed by Alur and Henzinger [7, 12].

Example 3 *Using* MTL, *we can write properties like*

$$\mathbf{G}\,(\textsf{problem} \rightarrow \mathbf{F}_{\leq 56}\,\textsf{alarm}) \qquad (2)$$

expressing that each time a problem *occurs, within* 56 *time units, an* alarm *rings.*

We can also express more involved properties, like

$$\mathbf{G}\,(\textsf{problem} \rightarrow (\mathbf{F}_{\leq 15}\,\textsf{repair} \vee \mathbf{G}_{[12,15]}\,\textsf{alarm}))$$

which expresses that each time a problem occurs, then either it is repaired in no more than 15 *time units, or an alarm rings for* 3 *time units* 12 *time units after the problem. There is no direct and obvious way to express this kind of property in* TCTL.

Remark 2 *The choice of the interpretation of* MTL *in terms of the pointwise or of the continuous semantics has a large impact on the meaning of the formulas, and as we will see later, also on their applicability in model-checking. The formula* $\mathbf{F}_{=2}\,a$ *expresses that an* a *will happen two time units later. In the continuous semantics, this formula is equivalent to* $\mathbf{F}_{=1}\,\mathbf{F}_{=1}\,a$ *(in one time unit, it will be the case that in one time unit, an* a *occurs). However, it is not the case in the pointwise semantics, as there may be no action one time unit later, hence any formula* $\mathbf{F}_{=1}\,\psi$ *would be evaluated as wrong from the initial point.*

Two extensions: TPTL and MTL+Past

In the following, we will also consider two extensions of MTL. First, as for TCTL, external clock variables can be used to express timing constraints. For instance, property (2) can be written as

$$\mathbf{G}\,(\textsf{problem} \rightarrow x.\mathbf{F}\,(\textsf{alarm} \wedge x \leq 56))$$

where x is a fresh variable which is reset when a problem occurs, and whose value is checked to be within $[0,56]$ when the alarm rings. The value of x is supposed to evolve at the same speed as the universal time (similar to a clock in a timed automaton). This logic with external clock variables is called TPTL, and has been first proposed in [7]. We give another example of formulas that can be expressed in TPTL:

$$\mathbf{G}\,(\textsf{problem} \rightarrow x.\mathbf{F}\,(\textsf{alarm} \wedge \mathbf{F}\,(\textsf{failsafe} \wedge x \leq 56))) \qquad (3)$$

This formula says that whenever a problem occurs, then within 56 time units, an alarm rings and later (but still within 56 time units since the problem occurred), the system enters a failsafe mode. It has been conjectured for a while (see [8, 10, 11, 36]), and proved in [19]: TPTL is strictly more expressive than MTL. For the pointwise semantics, formula (3) is a witness to that expressiveness result, meaning that formula (3) cannot be expressed in MTL. Surprisingly, this formula has an equivalent formula in MTL,[7] and a more involved formula has been proposed to distinguish between MTL and TPTL.

Following the classical untimed framework [40, 45], we also extend MTL with past-time modalities, *i.e.*, with the 'Since' modality, somewhat the dual of the 'Until' modality: formula $\varphi\,\mathbf{S}_I\,\psi$ expresses that φ holds since ψ was true (within I in the past). In the following, we will only use the simple formula $\mathbf{F}_I^{-1}\,\varphi$ which is the dual of $\mathbf{F}_I\,\varphi$ for the past: it expresses that φ was true in the past, within a delay belonging to the interval I. For instance, the formula

$$\mathbf{G}\,(a \rightarrow \mathbf{F}_{=1}^{-1}\,b)$$

expresses that every a is preceded one time unit earlier by a b. This logic will be denoted MTL+Past [9, 11]. Various expressiveness results comparing MTL+Past, MTL, TPTL can be found in the recent literature [19, 31, 32, 33]

4.2 The model-checking problem

The model-checking problem asks, given $\mathcal{A}$ a timed automaton and φ a formula, whether $\mathcal{A}$ satisfies φ, written $\mathcal{A} \models \varphi$, and meaning that all (accepting) runs of $\mathcal{A}$ satisfy the formula φ. Until very recently [48], MTL and TPTL were both considered as undecidable, as they both were able to express the propagating formula (2) [9, 11, 36]. However that was a bit misleading, as the decidability subtly depends on the choice of the semantics (being either pointwise or continuous) and on the fact that we consider infinite or finite executions in the timed automaton. In this subsection, we give the decidability results for those logics, clearly distinguishing between all important semantical parameters. We then give hints for understanding these (un)decidability results.

Theorem 2 *Over finite runs, the model-checking problem for:*

- MTL *under the pointwise semantics is decidable, and non-primitive recursive [50];*
- MTL *under the continuous semantics is undecidable [6];*
- MTL+Past *under the pointwise or the continuous semantics is undecidable;*

[7] Indeed, we can prove (*cf* [19]) that formula $x.\mathbf{F}\,(\textsf{alarm} \wedge \mathbf{F}\,(\textsf{failsafe} \wedge x \leq 56))$ is equivalent to

$$\begin{array}{ll} & \mathbf{F}_{\leq 28}\,\textsf{alarm} \wedge \mathbf{F}_{[28,56]}\,\textsf{failsafe} \\ \vee & \mathbf{F}_{\leq 28}\,(\textsf{alarm} \wedge \mathbf{F}_{\leq 28}\,\textsf{failsafe}) \\ \vee & \mathbf{F}_{\leq 28}\,(\mathbf{F}_{\leq 28}\,\textsf{alarm} \wedge \mathbf{F}_{=28}\,\textsf{failsafe}) \end{array}$$

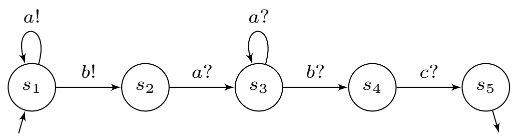
- TPTL *under the pointwise or the continuous semantics is undecidable [12].*

From these (un)decidability results, we learn that model-checking linear-time timed temporal logic is hard! And much harder than branching-time timed temporal logic. This is already the case in the untimed framework, but the gap dramatically increases in the timed framework. We first explain why it is so hard to model-check linear-time timed temporal logics. For that, we follow ideas developed in [27].

4.2.1 Model-checking linear-time timed properties is hard

We first explain the non-primitive recursive lower bound for the MTL model-checking problem, which relies on the halting problem for channel machines with insertion errors. A *channel machine* is a finite automaton which can write on a channel and read from it following a FIFO policy. We note '$a!$' for writing a at the tail of the channel and '$a?$' for reading an a at the head of the channel. A channel machine has insertion errors if any letter can be written at any time anywhere in the channel. A channel machine without insertion errors is said perfect, or insertion-free.

Example 4 *Consider the channel machine depicted on the next figure:*



A configuration of this system is a pair (s, w) where s is a discrete state of the machine and w is a word representing the content of the channel. We give an error-free computation example for that machine:

$$(s_1, \varepsilon) \xrightarrow{a!} (s_1, a) \xrightarrow{a!} (s_1, aa) \xrightarrow{b!} (s_2, aab) \xrightarrow{a?} (s_3, ab) \xrightarrow{a?} (s_3, b) \xrightarrow{b?} (s_4, \varepsilon)$$

We can see that no error-free computation allows to reach state s_5 (because no c is written on the channel). If we assume that this machine has insertion errors, then the following move is allowed:

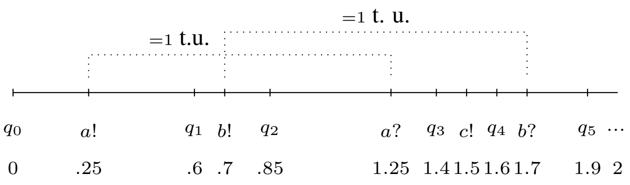
$$(s_4, \varepsilon) \xrightarrow{c?} (s_5, \varepsilon)$$

(we assume implicitely that a c has been inserted on the channel, so that the last transition labelled by '$c?$' can now be fired).

Given a channel machine $\mathcal{C}$ with a distinguished final state, the *halting problem* asks whether the machine $\mathcal{C}$ has an execution halting in the final state. The hardness results stated in Theorem 2 will be proved by reduction to the following problems about channel machines.

Proposition 1
- *The halting problem is undecidable for channel systems [25].*
- *The halting problem is non-primitive recursive for channel machines with insertion errors [53].*[8]

We now explain how MTL (and variants) can capture the behaviours of channel machines. The idea is to encode a computation of a channel machine as a timed word. In this encoding, the underlying untimed word is the trace of the computation, that is, an alternating sequence of states and actions. We use timing constraints to enforce the channel be FIFO: we require that any write action '$a!$' is followed one time unit later by a corresponding read action '$a?$'. This is not difficult to be convinced that this enforces the channel be FIFO. We illustrate this encoding on the next figure, which represents a timed word (actions and time stamps).



The above timed word encodes the following computation of the channel machine:

$$(q_0, \varepsilon) \xrightarrow{a!} (q_1, a) \xrightarrow{b!} (q_2, ab) \xrightarrow{a?} (q_3, b) \xrightarrow{c!} (q_4, bc) \xrightarrow{b?} (q_5, c) \cdots$$

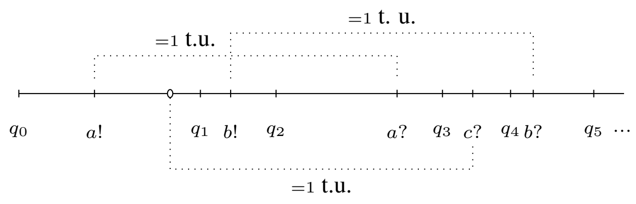
To properly encode a behaviour of a channel machine, a timed word must satisfy the following constraints:

- states and actions alternate. This can be checked using an LTL formula.
- the untimed projection of the timed words follows the rules of the channel machine. This can also be encoded with an LTL formula.
- the channel is FIFO: to do that we express that every write action is followed one time unit later by a corresponding read action. This can be expressed in MTL using formulas of the form:

$$\mathbf{G}\ (a! \rightarrow \mathbf{F}_{=1}\, a?)$$

  However, this formula does not encode the property that the channel behaves properly. Indeed, nothing prevents a read event '$a?$' to happen, even though there is no corresponding write event '$a!$' one time unit earlier. For instance, consider the following timed word:

[8] Formally, in [53], that's the halting problem for lossy channel machines which is proved non-primitive recursive, but this is indeed equivalent.



This timed word satisfies the propagating formulas $\mathbf{G}\ (a! \to \mathbf{F}_{=1}\, a?)$ (for every letter a), even though the event '$c?$' is not preceded by any action one unit earlier. The above formula hence only encodes the behaviour of a channel machine *with* insertion errors. However, from that study, we already learn that the model-checking of MTL (over finite words) is non-primitive recursive. To encode a perfect channel machine, we need to be able to express the property that every '$a?$' is preceded one time unit earlier by an '$a!$'. We call this property the 'backward matching property'.

We now discuss how we can express the backward matching property in timed temporal logics. Indeed, we would like to know whether MTL can express or not the behaviour of a perfect channel machine. We will present here natural ideas, which will happen to be wrong for MTL, but sufficient to prove undecidability of several variants or extensions of MTL.

- A first simple idea is to express this 'backward matching property' using the following formula:

$$\mathbf{G}\ ((\mathbf{F}_{=1}\, a?) \to a!)$$

  which expresses the fact that if there is a read event '$a?$' one time unit later, then there must be right now a corresponding write event '$a!$'. It is not hard to see that in the pointwise semantics, this does not express what we want. Indeed this formula is still satisfied by the above-mentioned timed word, because there is no action one time unit before the action '$c?$'. However, in the continuous semantics, this formula really enforces a perfect behaviour of the FIFO channel. That is why MTL in the continuous semantics has an undecidable model-checking problem.

- A second idea is to express this 'backward matching property' using a past-time modality (hence in MTL+Past). The formula

$$\mathbf{G}\ \left(a? \to \mathbf{F}^{-1}_{=1}\, a!\right)$$

  precisely expresses that every read event '$a?$' is preceded one time unit earlier by a matching write event '$a!$'. That is why MTL+Past is undecidable, even in the pointwise semantics.

- Finally, the 'backward matching property' can be expressed in TPTL using the following more involved property:

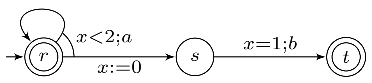
$$\neg\Big(\mathbf{F}\, x \cdot \mathbf{X}\, \big(y \cdot \mathbf{F}\, (x > 1 \wedge y < 1 \wedge a?)\big)\Big)$$

  Informally, this formula negates the fact that there are two consecutive positions (in the pointwise sense) such that an a is read more than one time unit after the first position, and less than time unit after the second position. This precisely negates the fact that there is an '$a?$' not preceded one time unit earlier by an action. This implies that TPTL is undecidable, already in the pointwise semantics (when at least two clock variables are used).

From all these considerations, we get that in the pointwise semantics, over finite runs, we can only express channel machines with insertion errors with MTL, whereas perfect channel machines can be expressed either using MTL in the continuous semantics, or using MTL+Past or TPTL in the pointwise semantics (both over finite words). This concludes the hardness results stated in Theorem 2.

4.2.2 MTL model-checking over finite words is decidable

We now explain how we can prove the decidability of MTL over finite words in the pointwise semantics. We know that LTL formulas can be transformed into alternating finite automata [47, 56]. In a similar way, we can transform any formula of MTL into an alternating timed automaton [43] with a single clock [48]. For instance, the formula $\mathbf{G}_{<2}\ \left(a\ \to\ \mathbf{F}_{=1}\, b\right)$ can be transformed into the following alternating timed automaton:



with the obvious interpretation that any time an a is done (within the two first time units), we fork a new thread which will check that a b appears one time unit later. A behaviour of this alternating timed automaton is an unbounded tree, and it is not obvious that it is possible to check for emptiness of such a system. Indeed, checking emptiness of alternating timed automata is decidable only for one clock over finite timed words, any slight extension (infinite timed words, two clocks, silent moves[9]) leading to undecidability [44, 50].

We explain how we can however understand the decidability of this model [48]. Consider the timed word $(c, 0.6)(a, 0.7)(a, 1.5)(b, 1.7)$. The execution of the above

[9]Or ε-transitions, if we follow the classical terminology in formal language theory.

alternating timed automaton on that timed word can be depicted as the following tree, which is not accepting as one of the branches (the second one on the picture) is not accepting (accepting states are underlined).

$$\begin{tikzcd}
\underline{r,0} \arrow[r, "c"] & \underline{r,0.6} \arrow[r, "a"] \arrow[dr] & \underline{r,0.7} \arrow[r, "a"] \arrow[dr] & \underline{r,1.5} \arrow[r, "b"] & \underline{r,1.7} \\
 & & s,0 \arrow[dr] & s,0 \arrow[r] & s,0.2 \\
 & & & s,0.8 \arrow[r] & \underline{t}
\end{tikzcd}$$

A configuration of the alternating timed automaton is a slice of the tree, for instance, $\{(r, 1.5), (s, 0), (s, 0.8)\}$ is a configuration. Because we consider finite words, there is no need to consider the tree structure of the execution, but we can reason globally on configurations of the automaton. There are infinitely many such configurations, but as for the region automaton construction for timed automata [4], the precise values of the clocks is not really relevant, and the things which are important in a configuration are the integral parts of the clocks and the relative order of the fractional parts of the clocks. For instance, for the above-mentioned configuration, we only need to know that there is a state $(s, 0)$ with fractional part 0, and two other states $(s, 0)$ and $(r, 1)$ such that the fractional part for $(s, 0)$ is greater than the fractional part for $(r, 1)$. For all configurations with the same abstraction, the possible future behaviours are the same, in a time-abstract bisimulation sense [50, 44]. Unfortunately, the set of abstractions of possible configurations of the alternating timed automaton is also infinite. The most important property is then that there is a *well-quasi-order* on the set of abstractions of configurations, and that we can use it to provide an algorithm to decide emptiness [34]! This briefly sketches an algorithm for deciding the MTL model-checking problem over finite timed words (in the pointwise semantics).

Note that we can prove the decidability of TPTL with a single internal variable applying the same method.

4.2.3 What about infinite behaviours?

All undecidability results we mentioned over finite timed words carry over into the framework of infinite timed words. Moreover, the algorithmic idea we presented to solve the model-checking problem for MTL over finite words cannot be lifted to the framework of infinite timed words, because checking emptiness of single-clock alternating timed automata over infinite timed words is undecidable [44]. Indeed, in that case, we can no more reason on slices of a tree execution, as we need to check the (say Büchi) acceptance condition on every branch of the tree. Moreover a construction *à la* Miyano-Hayashi [46] cannot be used, because the number of elements at a given level of an execution tree is potentially unbounded (because of the value of the clock), and it is not possible to use some well-quasi-order on slices as in the case of finite words to get a termination argument. Finally we get that we cannot circumvent the difficulty and find another tricky algorithm since we have the following undecidability result:

Theorem 3 *Over infinite timed words, the model-checking problem for* MTL *under the pointwise semantics is undecidable [49].*

4.3 Some interesting fragments of MTL

To encompass the high complexity of the model-checking of MTL, several fragments of MTL have been considered. We briefly describe some of those fragments, and give hints to explain why they are interesting and why they can be used for model-checking purposes.

4.3.1 The logic MITL

In 1991, the fragment MITL [10] has been proposed [5, 6], which basically disallows equalities for the constraints on the modalities. For instance, the formula $\mathbf{G}\ (a \rightarrow \mathbf{F}_{=1}\, b)$ is not in MITL, whereas $\mathbf{G}\ \left(a \rightarrow \mathbf{F}_{[1,2]}\, b\right)$ is in MITL. One reason why a positive result about MITL would be very satisfactory is that, in real timed systems, it is impossible to check a punctual constraint (because of the inherent imprecision of real systems). And disallowing punctual constraints leads indeed to an incredible improvement in the complexity of the model-checking problem!

Theorem 4 *Over infinite runs, the model-checking problem for* MITL *under the continuous semantics is* EXPSPACE-*Complete [6].*

The reason for this low[11] complexity is that the variability of a model of some property in MITL can be controlled, and any property expressible in MITL can be recognized by a timed automaton (hence is somehow very regular).

We give an example (taken out of [6]) to explain this regularity. Consider the MITL formula

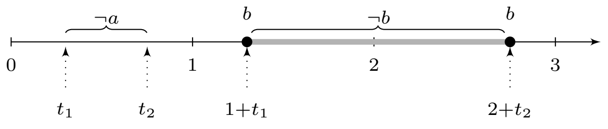
$$\varphi \;=\; \mathbf{G}_{(0,1)}\ (a \rightarrow \mathbf{F}_{[1,2]}\, b)$$

This formula says that within the first time unit, whenever a holds, then b must hold at some point between 1 and 2 time units later. To check the truth of this formula, a solution would be, each time a holds within the first time unit, to start a clock and check that within 1 to 2 time units from that point, b becomes true. However, there may be an unbounded number of a's within the first time unit. Hence, applying this method, an unbounded number of clocks would

[10]Standing for "Metric Interval Temporal Logic".

[11]In the context of timed systems, EXPSPACE is fairly low.

a priori be necessary. A more clever method needs to be used, which is illustrated below.



The idea is to point the last time b holds in the interval $[1, 2]$ and the first time b holds in the interval $[2, 3]$. Then, the only points within $(0, 1)$ that will not satisfy $\mathbf{F}_{[1,2]}\, b$ are in the interval (t_1, t_2), where $t_1{+}1$ is the point distinguished in $[1, 2]$ and $t_2 + 2$ is the point distiguished in $[2, 3]$. Hence, the only chance for the global formula to hold from the beginning is that there is no a in the interval (t_1, t_2). The strategy is then to build a timed automaton which guesses instants t_1 and t_2, resets clocks at those time points, and checks that those time points really satisfy the expected properties (for completeness, the constructed automaton is given on Figure 1). The algorithm developed in [6] is very involved, but relies on those kinds of ideas of smoothing the variability of behaviours w.r.t. MITL formulas, and build a timed automaton 'recognizing' the models of the formulas.

4.3.2 The logic Safety-MTL

The logic Safety-MTL has been proposed in [48] as a logic allowing to express safety properties like bounded response time properties of the form $\mathbf{G}\,(a \rightarrow \mathbf{F}_{=1}\, b)$ or $\mathbf{G}\,(a \rightarrow \mathbf{F}_{\leq 5}\,(b \wedge \mathbf{F}_{=1}\, c))$. Unlike MITL, it partly allows punctual constraints on modalities, but cannot express general response time properties. Roughly, a formula of MTL is in Safety-MTL whenever every positive instance[12] of an until modality is constrained by a bounded interval.

Theorem 5 *Over infinite runs, the model-checking problem for* Safety-MTL *in the pointwise semantics is decidable but non-primitive-recursive [48].*

The reason for the decidability of this logic is that it can only express 'safety' properties, that is properties whose negations, if violated, are violated after a finite prefix. For those formulas, we can build a single-clock timed alternating automaton over finite words and with only accepting states, which recognizes all bad prefixes of the formula. Then, the encoding for proving the decidability of alternating timed automata over finite timed words can be used in that case as well.

4.3.3 The logic coFlat-MTL

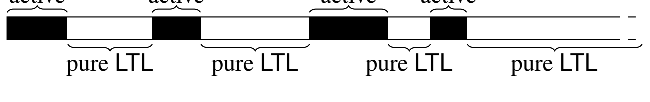
The syntax of the logic coFlat-MTL is not very intuitive [22], as it restricts the $\varphi\,\mathbf{U}_I\,\psi$-formulas in such a way that if I is not bounded, then ψ must be in LTL, whereas φ can be in coFlat-MTL itself. However, it is worth noticing that coFlat-MTL is rather powerful as it contains Bounded-MTL (the subset of MTL where all modalities are bounded), LTL, and is closed under invariance, which is rather useful for specifying correctness properties in critical systems. In particular, the formula $\mathbf{G}\ (a \rightarrow \mathbf{F}_{=1}\, b)$ is in coFlat-MTL, but $\mathbf{F}\,\mathbf{G}_{\leq 1}\, a$ is not in coFlat-MTL. Using technics completely different from those developed for MITL, we can prove the following result:

Theorem 6 *Over infinite runs, the model-checking problem for* coFlat-MTL *under the pointwise semantics is* EXPSPACE*-Complete [22].*

The decidability and upper bound are rather involved, and rely on channel machines. We have already mentioned that the halting problem for channel machines is undecidable (Proposition 1). However, here, we will not reduce to the general halting problem, but to the halting problem when we bound the number of cycles of the whole channel (this can be measured using an extra symbol which is immediately put on the channel when it is read; Then, the number of cycles corresponds to the number of times this symbol is read along a computation). This restricted problem is proved to be solvable in space polynomial in the size of the channel machine and in the value of the cycle bound [22].

We have already mentioned that we could encode MTL formulas into alternating timed automata. More precisely, we can encode the joint behaviours of a timed automaton and of a single-clock alternating timed automaton into a channel machine. In this encoding, a cycle of the channel corresponds to one time unit which has elapsed. If we consider a formula of Bounded-MTL (fragment of MTL where all modalities are bounded), it is rather clear that its verification only requires to look at a time-bounded prefix of the execution (the time bound can be set as the sum of all constants appearing in the formula). Hence, in the translation into channel machines, it means that we can restrict to channel machines with a bound on the number of cycles, which gives an algorithm for deciding the model-checking of Bounded-MTL.

We extend this idea to coFlat-MTL, and because of the syntactic restriction of coFlat-MTL, a timed word not satisfying a formula φ in coFlat-MTL can be decomposed as follows:



where the number of active fragments is at most exponential and the total duration of active fragments is also at most

[12] *I.e.*, every instance under the scope of an even number of negations.

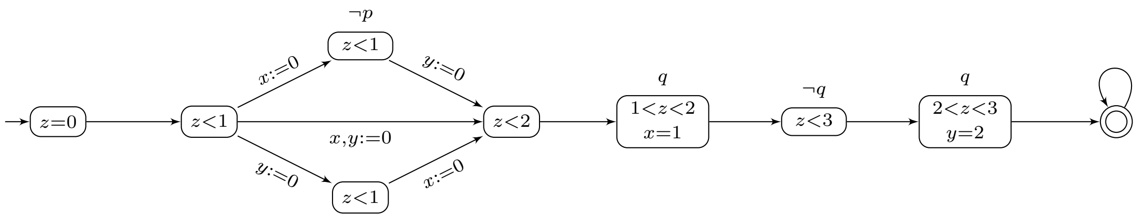


Figure 1. The automaton for formula $\varphi = \mathbf{G}_{(0,1)}\,(a \rightarrow \mathbf{F}_{[1,2]}\,b)$

exponential. An active fragment corresponds to a cycle-bounded computation in a channel machine, whereas the pure LTL parts are very simple computations corresponding to those of a finite automaton. We do not enter into more details here.

Remark 3 *It is worth noticing here that, unlike* MITL, coFlat-MTL *does not only express 'regular' properties. For instance, as the property* $\mathbf{G}\,(a \rightarrow \mathbf{F}_{=1}\,b)$ *belongs to* coFlat-MTL, *we can easily generate the context-free language* $\{a^n b^m \mid m \geq n\}$ *with* coFlat-MTL.

We have very recently mixed the MITL and the coFlat-MTL approaches to obtain a very expressive fragment of MTL (more precisely a fragment corresponding to coFlat-MTL, except that the restriction to LTL formulas is replaced by a restriction to MITL). For that fragment of MTL and in the continuous semantics, the model-checking problems (over infinite runs) is also EXPSPACE-Complete [23].

5 Conclusion

In this paper, we have presented several formalisms to specify real-time properties of systems. We have discussed their applicability to model-checking by giving their decidability and complexity. The branching-time logic TCTL has a rather low complexity, and is indeed implemented in the tool Kronos [24]. Though our aim is not to discuss the eternal dilemna between branching-time and linear-time (for that we better refer to [55]), the linear-time timed temporal logics MTL and variants seem more powerful, as they can express multiple timing constraints on runs, which is really interesting for writing specifications.

Those linear-time timed temporal logics have been defined in the beginning of the 90s. Several works have been done at that time, but since few years, these logics have raised a new interest in the community. And even though the full logics are expensive (if not undecidable), several large fragments have been considered, whose model-checking is rather cheap! Right now, there is no tool implementing those logics, but we suspect that it could be really interesting to develop new data structures and algorithms for verifying these logics.

Recently, new challenges have risen, in which time is not sufficient to express quantitative constraints on real systems. Indeed, one can be interested in expressing energy consumption constraints, constraints on costs, bandwidth resources, *etc*. For that, the model of priced (or weighted) timed automata has been proposed [13, 15] which extends timed automata with observer variables and allows to model those new kinds of constraints. Logics have thus been extended to express those constraints, and everything becomes even much harder... See for instance [26, 18, 17, 20, 21].

Acknowledgements: I want to thank all the persons I have recently worked with on timed temporal logics, Fabrice Chevalier, Nicolas Markey, Joël Ouaknine, and James Worrell.

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the 5th Annual Symposium on Logic in Computer Science (LICS'90)*, pages 414–425. IEEE Computer Society Press, 1990.

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

[3] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

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

[5] R. Alur, T. Feder, and Th. A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the 10th Annual ACM Symposium on Princiles of Distributed Computing (PODC'91)*, pages 139–152. ACM, 1991.

[6] R. Alur, T. Feder, and Th. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[7] R. Alur and Th. A. Henzinger. A really temporal logic. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*, pages 164–169. IEEE Computer Society Press, 1989.

[8] R. Alur and Th. A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proceedings of the 5th Annual Symposium on Logic in Computer Science (LICS'90)*, pages

390–401. IEEE Computer Society Press, 1990. Preliminary version of [11].

[9] R. Alur and Th. A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS'92)*, pages 177–186. IEEE Computer Society Press, 1992.

[10] R. Alur and Th. A. Henzinger. Logics and models of real-time: A survey. In *Real-Time: Theory in Practice, Proceedings of the REX Workshop 1991*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer, 1992.

[11] R. Alur and Th. A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[12] R. Alur and Th. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[13] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2001.

[14] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems (QEST'06)*, pages 125–126. IEEE Computer Society Press, 2006.

[15] G. Behrmann, A. Fehnker, Th. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

[16] H. Bel mokadem, B. Bérard, P. Bouyer, and F. Laroussinie. A new modality for almost everywhere properties in timed automata. In *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2005.

[17] P. Bouyer. Weighted timed automata: Model-checking and games. In *Proceedings of the 22nd Conf. Mathematical Foundations of Programming Semantics (MFPS XXII)*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 3–17. Elsevier, 2006. Invited paper.

[18] P. Bouyer, Th. Brihaye, and N. Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, 2006.

[19] P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of TPTL and MTL. In *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 432–443. Springer, 2005.

[20] P. Bouyer, K. G. Larsen, and N. Markey. Model-checking one-clock priced timed automata. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'07)*, volume 4423 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2007.

[21] P. Bouyer and N. Markey. Costs are expensive! In *Proceedings of the 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2007.

[22] P. Bouyer, N. Markey, J. Ouaknine, and J. Worrell. The cost of punctuality. In *Proceedings of the 22nd Annual Symposium on Logic in Computer Science (LICS'07)*, pages 109–118. IEEE Computer Society Press, 2007.

[23] P. Bouyer, N. Markey, J. Ouaknine, and J. Worrell. On expressiveness and complexity in real-time model checking. Submitted, 2008.

[24] 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 the 10th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998.

[25] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[26] Th. Brihaye, V. Bruyère, and J.-F. Raskin. Model-checking for weighted timed automata. In *Proceedings of the Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, volume 3253 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2004.

[27] F. Chevalier. *Logiques pour les systèmes temporisés : contrôle et expressivité*. PhD thesis, École Normale Supérieure de Cachan, France, 2007.

[28] F. Chevalier, D. D'Souza, and P. Prabhakar. Counter-free input determined timed automata. In *Proceedings of the 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2007.

[29] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[30] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Proceedings of the Hybrid Systems III: Verification and Control (1995)*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1996.

[31] D. D'Souza and P. Prabhakar. On the expressiveness of MTL with past operators. In *Proceedings of the 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2006.

[32] D. D'Souza and P. Prabhakar. On the expressiveness of MTL in the pointwise and continuous semantics. *International Journal on Software Tools for Technology Transfer*, 9(1):1–4, 2007.

[33] D. D'Souza, M. Raj Mohan, and P. Prabhakar. Eliminating past operators in Metric Temporal Logic. Manuscript, 2007.

[34] A. Finkel and P. Schnoebelen. Well structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[35] K. Havelund, A. Skou, K. G. Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: An

industrial case study using uppaal. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 2–13. IEEE Computer Society Press, 1997.

[36] Th. A. Henzinger. It's about time: Real-time logics reviewed. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 439–454. Springer, 1998.

[37] Th. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model-checker for hybrid systems. *Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

[38] Th. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[39] Y. Hirshfeld and A. Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62(1):1–28, 2004.

[40] J. A. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, Los Angeles, CA, USA, 1968.

[41] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[42] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

[43] S. Lasota and I. Walukiewicz. Alternating timed automata. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2005.

[44] S. Lasota and I. Walukiewicz. Alternating timed automata. *ACM Transactions on Computational Logic*, 2007. To appear.

[45] O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Proceedings of the Conference on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 413–424. Springer-Verlag, 1985.

[46] S. Miyano and T. Hayashi. Alternating finite automata on omega-words. *Theoretical Computer Science*, 32:321–330, 1984.

[47] D. E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings of the 3rd Annual Symposium on Logic in Computer Science (LICS'88)*, pages 422–427. IEEE Computer Society Press, 1988.

[48] J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *Proceedings of the 19th Annual Symposium on Logic in Computer Science (LICS'05)*, pages 188–197. IEEE Computer Society Press, 2005.

[49] J. Ouaknine and J. Worrell. On metric temporal logic and faulty Turing machines. In *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.

[50] J. Ouaknine and J. Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, 3(1:8):1–27, 2007.

[51] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[52] J.-F. Raskin. *Logics, Automata and Classical Theories for Deciding Real-Time*. PhD thesis, Université de Namur, Belgium, 1999.

[53] Ph. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.

[54] P.-Y. Schobbens, J.-F. Raskin, and T. A. Henzinger. Axioms for real-time logics. *Theoretical Computer Science*, 274(1–2):151–182, 2002.

[55] M. Vardi. Branching *vs* linear time: Final showdown. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001. Invited talk.

[56] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proceedings of the Logics for Concurrency: Structure versus Automata (1995)*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1996.