Is your Model Checker on Time?

On the Complexity of Model Checking for Timed Modal Logics

Luca Aceto[1] and François Laroussinie[2]

[1] BRICS***, Department of Computer Science, Aalborg University,
Fredrik Bajers Vej 7-E, DK-9220 Aalborg Ø, Denmark.
Email: `luca@cs.auc.dk`, Fax: +45 98 15 98 89

[2] Laboratoire Spécification et Vérification, CNRS UMR 8643,
ENS de Cachan, 61 av. du Pr. Wilson, 94235 Cachan, France.
Email: `fl@lsv.ens-cachan.fr`, Fax: +33 1 47 40 24 64

Abstract. This paper studies the structural complexity of model checking for (variations on) the specification formalisms used in the tools CMC and UPPAAL, and fragments of a timed alternation-free μ-calculus. For each of the logics we study, we characterize the computational complexity of model checking, as well as its specification and program complexity, using timed automata as our system model.

1 Introduction

The extension of the model checking paradigm to the specification and verification of real-time systems has been thoroughly studied in the last few years. This extensive research effort has led to the development of specification logics that extend standard untimed formalisms with the quantitative analysis of timing constraints (see, e.g., [4, 15, 18]), and to important theoretical results setting the limits of decidability for model checking. This theory is now embodied in verification tools like HyTech [23], Kronos [24] and UPPAAL [20], which have been successfully applied to the verification of real sized systems.

The successful application of the aforementioned verification tools to the analysis of realistic systems indicates that automatic verification of real-time, embedded software may be feasible in practice. However, despite many important theoretical results presented in *op. cit.*, the literature is lacking a comprehensive analysis of the structural complexity of model checking for real-time logics. In the untimed case, model checking algorithms with a polynomial time complexity, and often small space requirements, have been developed for several branching time temporal logics [8, 9]. In the timed case, most of the model checking problems considered in the literature are PSPACE-hard [3, 10, 15]. Clearly the quantitative analysis of timing constraints increases the complexity of model checking, but it is interesting to analyze precisely in which cases this complexity blow-up occurs. In the untimed case, several papers (see, e.g., [13, 22, 11]) study in detail the effect of the temporal operators, the number of atomic propositions

*** Basic Research in Computer Science.

or the depth of operators' nesting in the complexity of model checking, giving a better understanding of the complexity issue. Here, among other things, we address the same kind of problem for the timed case: what happens if time is inserted either only in the model or only in the formula? And what happens if we use less expressive logics with restricted operators?

We consider several timed modal logics: L_ν has been introduced in [18], and is the specification language used in the tool CMC [17]; L_s is a fragment of L_ν which has been proposed in [19] in order to improve the efficiency of model checking in practice; SBLL [2] and $L_{\forall S}$ [1] have been introduced for their properties w.r.t. the testing timed automaton method that is currently used in verification tools like UPPAAL to check for properties other than plain reachability ones.

For each of these property languages, we study the computational complexity of model checking, using timed automata [5] as our system model. As argued by Lichtenstein and Pnueli [21], the complexity of the model checking problem can be measured in three different ways. First, one can fix the specification and measure the complexity as a function of the size of the program being verified (the *program complexity* measure). Secondly, one can fix the program and measure the complexity as a function of the size of the specification (the *specification complexity* measure). Finally, the combined complexity of the model checking problem is measured as a function of the size of both the program and the specification. In this paper we offer complexity results for these three different views of the model checking problem for the logics we consider. In so doing, we give an *a posteriori* justification, couched in complexity-theoretic arguments, for some of the folk beliefs in the area of model checking for real-time systems, and for some of the choices made by developers of real-time verification tools.

Outline of the Main Results. We begin by analyzing the complexity of model checking for $\mathrm{L}_{\mu,\nu}$, a timed alternation-free modal μ-calculus (AFMC). In the untimed setting, such a fragment of the modal μ-calculus plays an important role as a specification formalism because it is fairly expressive and its restricted syntax makes the symbolic evaluation of expressions very simple (more precisely, linear both in the size of the model and the specification). In the real-time setting, we show that the complexity of model checking for the timed AFMC, and for its sublogic L_ν, is EXPTIME-complete, as are both the program complexity and the specification complexity. (Perhaps surprisingly, the model checking problem for L_ν—and *a fortiori* for the timed AFMC—is EXPTIME-complete even if we fix the model to be the inactive process without clocks, *nil*.) We also prove that the model checking problem for L_ν without greatest fixpoints—essentially, a timed version of Hennessy-Milner logic [14]—is PSPACE-complete.

It is instructive to compare the above results with similar ones for the untimed alternation-free μ-calculus. As previously mentioned, for such a program logic, we have algorithms for model checking that run in time linear both in the size of the program and of the specification. Moreover, both the program and the specification complexities are P-complete [6, 12]. Note, however, that the program complexity of the alternation-free μ-calculus for *concurrent* programs is EXPTIME-complete [6], and this matches exactly the complexity results we

	Model checking	Prog compl.	Spec compl.
$\mathrm{L}_{\mu,\nu}, \mathrm{L}_{\nu}$	EXPTIME-complete	EXPTIME-complete	EXPTIME-complete
$L_s, SBLL, L_{\forall S}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
L_{ν}^{-}	PSPACE-complete	P	PSPACE-complete
L_s^{-}	coNP-complete	P	coNP-complete
$SBLL^{-}, L_{\forall S}^{-}$	PSPACE-complete	PSPACE-complete	coNP-complete

Table 1: Overview of the Results

offer for $\mathrm{L}_{\mu,\nu}$ model checking. It is also interesting to note that the complexity of CTL model checking and reachability for concurrent programs is PSPACE-complete [6], matching the complexity of model checking for TCTL [4] and of reachability in timed automata, respectively. These results seem to provide a mathematical grounding to the folk belief that "clocks act like concurrent programs", and that increasing the number of clocks corresponds to adding parallel components.

We then proceed to develop a thorough analysis of the complexity of model checking for all the other timed modal property languages that we have found in the literature. In each case, we offer results pinpointing the program, the specification as well as the combined complexity of model checking for the property languages with and without fixpoints. An overview of the results we have obtained is presented in Table 1, where L^{-} denotes the fixpoint free fragment of L. Here we just wish to point out that the model checking problem for the property language L_s is PSPACE-complete, no matter whether the complexity is measured with respect to the size of the program, of the specification or of both. In light of the aforementioned results, and assuming that PSPACE is different from EXPTIME, the model checking problem for L_s has a lower computational complexity than that for L_{ν}. Our results thus offer a complexity-theoretic justification for the claims in [19]. The source of the lower complexity derives from the observation that the model checking problem for L_s, unlike that for L_{ν}, can be reduced in polynomial time to reachability checking in timed automata—a problem whose PSPACE-completeness was shown in [5].

2 Basic definitions

We begin by briefly reviewing a variation on the timed automaton model proposed by Alur and Dill [5] and the property languages that will be used in this study.

Timed Automata. Let Act be a finite set of *actions*, and let $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ denote the sets of natural and non-negative real numbers, respectively. We write $\mathcal{D}$ for the set of *delay actions* $\{\epsilon(d) \mid d \in \mathbb{R}_{\geq 0}\}$.

Let C be a set of clocks. We use $\mathcal{B}(C)$ to denote the set of boolean expressions over atomic formulae of the form $x \sim p$ and $x - y \sim p$, with $x, y \in C$, $p \in \mathbb{N}$, and $\sim \in \{<, >, =\}$. Moreover we write $\mathcal{B}_k(C)$ for the restriction of $\mathcal{B}(C)$ to expressions where the integer constants belong to $\{0, \ldots, k\}$. Expressions in $\mathcal{B}(C)$ are interpreted over the collection of time assignments. A *time assignment*, or *valuation*, v for C is a function from C to $\mathbb{R}_{\geq 0}$. We write $\mathbb{R}_{\geq 0}^{C}$ for the collection of valuations for C. Given $g \in \mathcal{B}(C)$ and a time assignment v, the boolean value $g(v)$ describes whether g is satisfied by v or not. For every time assignment v and $d \in \mathbb{R}_{\geq 0}$, we use $v+d$ to denote the time assignment which maps each clock $x \in C$ to the value $v(x)+d$. For every $C' \subseteq C$, $[C' \to 0]v$ denotes the assignment for C which maps each clock in C' to the value 0 and agrees with v over $C \backslash C'$.

Definition 1. *A* timed automaton *(TA) is a quintuple* $A = \langle \mathsf{Act}, N, n_0, C, E \rangle$ *where* N *is a finite set of* nodes, n_0 *is the* initial node, C *is a finite set of* clocks, *and* $E \subseteq N \times \mathcal{B}(C) \times \mathsf{Act} \times 2^C \times N$ *is a set of* edges. *The tuple* $e = \langle n, g, a, r, n' \rangle \in E$ *stands for an edge from node* n *to node* n' *with action* a*, where* r *denotes the set of clocks to be reset to* 0 *and* g *is the enabling condition (or* guard*). We use* $\mathtt{MCst}(A)$ *to denote the largest integer constant occurring in the guards of* A*.*

A *state* (or configuration) of a timed automaton A is a pair (n, v) where n is a node of A and v is a time assignment for C. The initial state of A is $(n_0, [C \to 0])$ where n_0 is the initial node of A, and $[C \to 0]$ is the time assignment mapping all clocks in C to 0. The operational semantics of a timed automaton A is given by the Timed Labelled Transition System (TLTS) $\mathcal{T}_A = \langle \mathcal{S}_A, \mathsf{Act} \cup \mathcal{D}, s^0, \longrightarrow \rangle$, where $\mathcal{S}_A$ is the set of states of A, s^0 is the initial state of A, and $\longrightarrow$ is the transition relation defined as follows:

$$(n, v) \xrightarrow{a} (n', v') \text{ iff } \exists \langle n, g, a, r, n' \rangle \in E.\ g(v) = \mathtt{tt} \wedge v' = [r \to 0]v$$

$$(n, v) \xrightarrow{\epsilon(d)} (n', v') \text{ iff } n = n' \text{ and } v' = v + d$$

Remark 1. Note that we could consider *extended TAs* where we assign an *invariant* (i.e. a downward closed clock constraint) to each node to avoid excessive time delays. All the results presented here will still hold for extended TAs. Note that, given a complexity class $\mathcal{C}$, having a $\mathcal{C}$-hardness result for (simple) TAs implies the same for extended TAs, while having a $\mathcal{C}$ membership result for extended TAs implies the same for TAs.

The specification languages. We now define $\mathrm{L}_{\mu,\nu}$ a timed alternation-free modal μ-calculus.

Definition 2. *Let* K *be a finite set of clocks,* Id *a set of identifiers. The set* $\mathrm{L}_{\mu,\nu}$ *of formulae over* K *and* Id *is generated by the following grammar:*

$$\begin{aligned} \mathrm{L}_{\mu,\nu} \ni \psi, \varphi \ ::= \ & g \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle a \rangle\, \varphi \mid [a]\, \varphi \mid \exists \varphi \mid \forall \varphi \\ & \mid K' \ \underline{\mathtt{in}}\ \varphi \mid \mathtt{max}(X, \varphi) \mid \mathtt{min}(X, \varphi) \mid X \end{aligned}$$

where $a \in \mathsf{Act}$*,* $g \in \mathcal{B}(K)$*,* $K' \subseteq K$ *and* $X \in \mathsf{Id}$*. Moreover, each occurrence of an identifier* X *in a formula has to be bound by a* $\mathtt{min}(X, \varphi)$ *(or* $\mathtt{max}(X, \varphi)$*) operator, and it cannot occur in a* φ*-subformula of the form* $\mathtt{max}(X', \psi)$ *(resp.* $\mathtt{min}(X', \psi)$*). (This restriction corresponds to the "alternation-free" property.)*

New operators like $\mathtt{tt}$, $\mathtt{ff}$, $g \Rightarrow \psi$ (read 'g implies ψ') can be easily defined. Let $\mathtt{MCst}(\varphi)$ be the largest integer constant occurring in the clock constraints in φ. Given a TA A, we interpret formulae in $\mathrm{L}_{\mu,\nu}$ w.r.t. *extended configurations*

$$
\begin{array}{rcl}
(n,v,u) \models [a]\,\varphi & \text{iff} & \forall\,(n',v').\ (n,v) \xrightarrow{a} (n',v') \ \Rightarrow\ (n',v',u) \models \varphi \\
(n,v,u) \models \forall\!\!\!\forall\,\varphi & \text{iff} & \forall d \in \mathbb{R}_{\geq 0},\ (n,v+d,u+d) \models \varphi \\
(n,v,u) \models g & \text{iff} & g(u) = \mathtt{tt} \\
(n,v,u) \models K'\ \underline{\mathtt{in}}\ \varphi & \text{iff} & (n,v,[K' \to 0]u) \models \varphi \\
(n,v,u) \models \mathtt{max}(X,\varphi) & \text{iff} & (n,v,u) \text{ belongs to the largest solution of } X = \varphi
\end{array}
$$

Table 2: Semantics of $\mathrm{L}_{\mu,\nu}$.

(n,v,u), where (n,v) is a configuration of A and u is a time assignment for K. Whereas the classical modal operators $\langle a\rangle$ and $[a]$ deal with action transitions, the operator $\exists\!\!\!\exists$ (resp. $\forall\!\!\!\forall$) denotes existential (resp. universal) quantification over delay transitions. The clocks in K are so-called formula clocks; they increase synchronously with the automata clocks, and they are used as stopwatches for measuring the time elapsing between states of the system. The formula $K'\ \underline{\mathtt{in}}\ \varphi$ initializes the set of formula clocks K' to 0 in φ. The constraints g are used to compare the value of formula clocks in the current extended configuration with an integer value. Finally, an extended configuration satisfies $\mathtt{max}(Z,\varphi)$ (resp. ($\mathtt{min}(Z,\varphi)$) if it belongs to the largest (resp. least) solution of the equation $Z = \varphi$ over the complete lattice of sets of extended configurations. The existence of these solutions is guaranteed by standard fixpoint theory. The semantics of $\mathrm{L}_{\mu,\nu}$ is sketched in Table 2. (The operators $\langle a\rangle$ and $\exists\!\!\!\exists$ are duals of $[a]$ and $\forall\!\!\!\forall$; the semantics of boolean operators is omitted.) The full formal details of the semantics are standard [16].

As an example of a property that can be expressed in $\mathrm{L}_{\mu,\nu}$ using fixpoints and clock constraints, consider the formula

$$\mathtt{max}\Big(X, \Big([b]\{x\}\ \underline{\mathtt{in}}\ \exists\!\!\!\exists(\langle c\rangle\, \mathtt{tt} \wedge x \leq 3)\Big) \wedge [a]X \wedge \forall\!\!\!\forall X\Big) \ .$$

This formula expresses the fact that, in every state that is reachable by performing a-actions and delays, every occurrence of a b-action can be followed by a c-action within 3 time units.

Fragments of $\mathrm{L}_{\mu,\nu}$. The logic L_ν [18] is the fragment of $\mathrm{L}_{\mu,\nu}$ in which only greatest fixpoints are allowed. The logic L_s [19] is the fragment of L_ν without the existential modalities $\langle a\rangle$ and $\exists\!\!\!\exists$, and where only a restricted disjunction of the form $g \vee \varphi$ (with $g \in \mathcal{B}(K)$) is allowed.

The property languages $SBLL$ and $L_{\forall S}$ extend L_s, and use a slightly different kind of TAs where (1) $\mathcal{U}$ is a subset of Act s.t. any edge labeled with $a \in \mathcal{U}$ has the guard $\mathtt{tt}$ and (2) Act contains the label τ used for the internal action of automata. Moreover they are based on different semantics (denoted by $\vdash$) compared with L_ν and L_s: a formula φ holds for (n,v,u) only if φ holds for every (n',v',u) with (n',v') reachable from (n,v) in zero or more τ-transitions. For example, $(n,v,u) \vdash [a]\varphi$ iff for every $(n,v) \xrightarrow{\tau}^{*} (n',v')$ we have that $(n',v') \xrightarrow{a} (n'',v'')$ implies $(n'',v'',u) \vdash \varphi$. Moreover $(n,v,u) \vdash \forall\!\!\!\forall\varphi$ iff for every (n',v') reached from (n,v) by using τ-transitions and delay transitions (of total duration d), we have $(n',v',u+d) \vdash \varphi$.

$SBLL$ extends L_s by allowing the use of $\langle a\rangle\,\mathtt{tt}$ subformulae with $a \in \mathcal{U}$. $L_{\forall S}$

extends $SBLL$ with new operators $\forall\!\!\!\forall_S$ with $S \subseteq \mathcal{U}$. A formula $\forall\!\!\!\forall_S \varphi$ holds for (n, v, u) iff φ holds for any $(n', v', u + d)$ s.t. (n', v') is reachable from (n, v) by using only τ-transitions and delay transitions (with total duration d), but delay transitions occur only in states in which none of the actions in S are enabled.
These two languages can be translated into L_ν in the following sense: for any $\varphi \in L_{\forall S}$, there exists an L_ν formula $\overline{\varphi}$ s.t. $A \vdash \varphi$ iff $A \models \overline{\varphi}$. For example, we have that $\overline{[a]\psi} = \mathtt{max}(X, [a]\overline{\psi} \wedge [\tau]X)$. An important property [2, 1] of $SBLL$ and $L_{\forall S}$ is that their model checking problem can be reduced to a reachability problem: for any formula φ of these languages, we can build a *testing automaton* T_φ s.t. $A \vdash \varphi$ iff a reject node is not reachable in the parallel composition $(A|T_\varphi)$. Moreover it has been shown that $L_{\forall S}$ is expressive enough to encode any reachability property [1].

Verification of timed systems. Automatic verification of timed systems is possible despite the uncountably infinite number of configurations associated with a timed automaton. The decision procedure for $A \models \varphi$ is based on the well known *region technique* [4]. Given A and φ, it is possible to partition the infinite set of time assignments over $C^+ = C \cup K$ into a finite number of regions in such a way that two extended configurations (n, u) and (n, v), where $u, v \in \mathbb{R}_{\geq 0}^{C^+}$ are in the same region, satisfy the same formulae. Formally the regions can be defined as the equivalence classes induced by the equivalence relation over valuations defined thus: given $u, v \in \mathbb{R}_{\geq 0}^{C^+}$, u and v are in the same region iff they satisfy the same clock constraints in $\mathcal{B}_M(C^+)$, where $M = \max(\mathtt{MCst}(A), \mathtt{MCst}(\varphi))$.
We write $[u]$ for the region which contains the time assignment u, and use $\mathcal{R}_k^{Cl}$ to denote the (finite) set of all regions for a set Cl of clocks and the maximum constant k. Given a region $[u]$ in $\mathcal{R}_k^{Cl}$ and $C' \subseteq Cl$, we define the reset operator thus: $[C' \to 0][u] = [[C' \to 0]u]$. Moreover, given a region γ, its successor region, denoted by $succ(\gamma)$, is the region γ' s.t. for any $u \in \gamma$ there exists $\delta \in \mathbb{R}_{\geq 0}$ with $[u+\delta] = \gamma'$, and $[u+\delta'] \in \{\gamma, \gamma'\}$, for every $\delta' < \delta$. The region $succ(\gamma)$ is different from γ iff $\gamma(x \leq k) = \mathtt{tt}$ for some clock x.
Now, given a timed automaton $A = \langle \mathsf{Act}, N, n_0, C, E \rangle$, a set K of formula clocks and an integer constant M with $M \geq \mathtt{MCst}(A)$, we can define a symbolic semantics [18] over the finite transition system $(S, \to)$, called *region graph*, defined thus: $S = N \times \mathcal{R}_M^{C \cup K}$ and $\to = (\bigcup_a \xrightarrow{a}) \cup \xrightarrow{succ}$. The symbolic semantics is closely related to the standard one: for every $L_{\mu,\nu}$ formula whose clock constraints do not use constants greater than M, and $u \in \gamma$, $(n, \gamma) \models \varphi$ iff $(n, u) \models \varphi$. Therefore each instance of the timed model checking problem can be reduced to an untimed model checking query over the region graph. Note that the size of $\mathcal{R}_M^{C^+}$ is in $O(|C^+|! \cdot M^{|C^+|})$. Moreover for any region γ, $|\{\gamma' | \gamma' = succ^i(\gamma), i \in \mathbb{N}\}| \leq 2 \cdot |C^+| \cdot (M + 1)$.
The reachability problem, which is a fundamental question in system verification, is know to be PSPACE-complete [3, 10]. Moreover the model checking problem for TCTL (a timed extension of CTL) is PSPACE-complete [4].

We shall use the abbreviations $\mathcal{A}_{+t} \stackrel{?}{\models} \Psi_{+t}$, $\mathcal{A} \stackrel{?}{\models} \Psi_{+t}$ and $\mathcal{A}_{+t} \stackrel{?}{\models} \Psi$ to denote, respectively, the model checking problems where clocks are allowed both in automata and specifications, where clocks are allowed only in specifications and where clocks are allowed only in automata.

3 Complexity results for model checking

We now consider the complexity of model checking (MC) for the property languages introduced previously. These results require to define what are the size of a timed automaton $A = \langle \mathsf{Act}, N, n_0, C, E\rangle$ and a formula $\varphi \in \mathrm{L}_{\mu,\nu}$. The size $|\varphi|$ of a formula is its length. We define $|A|$ as $|N| + |C| + |E| + \mathtt{MCst}(A) + \Sigma_{e\in E}|g_e|$, and assume a binary encoding for the elements of the sets N and C. Considering constants represented in unary or binary does not change our results except when it is explicitly mentioned.

Theorem 1. *The complexity of* $\mathrm{L}_{\mu,\nu}$ *and* L_{ν} *model checking is EXPTIME-complete. Moreover, we have that the specification and program complexities of* $\mathrm{L}_{\mu,\nu}$ *and* L_{ν} *model checking are also EXPTIME-complete.*

Proof. **EXPTIME membership:** We have seen that $A \models \varphi$ iff $\tilde{A} \models \varphi$ where $\tilde{A}$ is an untimed automaton (the *region graph*) whose size is exponential in $|A|$ and over which φ is interpreted as an untimed formula. If we modify slightly $\tilde{A}$ by adding the transitive closure of $\xrightarrow{succ}$, the size of the resulting automaton is still exponential in $|A|$, and $\exists\!\!\!\exists$ and $\forall\!\!\!\forall$ become "one step" modalities. Then φ is a simple (untimed) alternation-free μ-calculus formula for which model checking is linear in $|\tilde{A}|$ and $|\varphi|$ [9]. This gives the EXPTIME membership for $\mathrm{L}_{\mu,\nu}$ and L_{ν}.
EXPTIME-hardness: Deciding whether a given linear bounded alternating Turing machine (LBATM) $\mathcal{M}$ accepts a given input string w is EXPTIME-complete [7], and it can be reduced in polynomial time to a MC problem $A_{\mathcal{M}} \models \Phi$ with $\Phi \in \mathrm{L}_{\nu}$. The main idea is that we can build a TA $A_{\mathcal{M}}$ over actions s and `accept` s.t. any s-transition of $A_{\mathcal{M}}$ corresponds to a step of $\mathcal{M}$ due to the tape boundness (see [3, 10]). By following the same approach proposed in [6] for untimed concurrent systems, the alternating behaviour[1] of $\mathcal{M}$ can be handled by an L_{ν} formula of the form: $\Phi = \mathtt{max}(X, [\mathtt{accept}]\mathtt{ff} \wedge \forall\!\!\!\forall\ [s]\ \exists\!\!\!\exists\ \langle s\rangle\, X)$. Intuitively Φ holds for $A_{\mathcal{M}}$ if the current "or" state is not an accepting state and after any step (leading to an "and" state), there exists a transition leading to a non-accepting "or" state and so on. We have $A_{\mathcal{M}} \models \Phi$ iff the LBATM $\mathcal{M}$ does not accept w. This gives the EXPTIME-hardness for L_{ν} and $\mathrm{L}_{\mu,\nu}$.
Specification complexity: In fact the acceptance of w by a LBATM $\mathcal{M}$ can be reduced in polynomial time to a problem of the form $nil \models \Psi_{\mathcal{M},w}$ where $\Psi_{\mathcal{M},w}$ is an L_{ν} formula. This encoding is based on the use of formula clocks to represent the configurations of $\mathcal{M}$. This gives the EXPTIME-hardness.
Program complexity: This is due to the proof of EXPTIME-hardness for L_{ν} model checking where the formula $\Phi = \mathtt{max}(X, [\mathtt{accept}]\mathtt{ff} \wedge \forall\!\!\!\forall\ [s]\ \exists\!\!\!\exists\ \langle s\rangle\, X)$ does not depend on the LBATM $\mathcal{M}$. □

[1] We assume w.l.o.g. that we have a strict alternation of "or" and "and" states in $\mathcal{M}$, and that the initial and final states are "or" states.

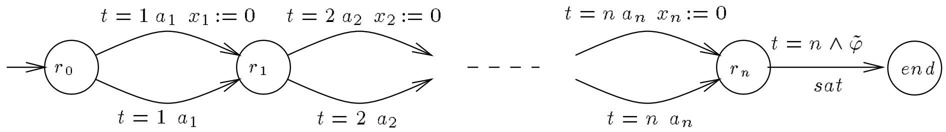

Fig. 1: The automaton $A_{(\varphi)}$ with $\tilde{\varphi} = \varphi[p_i \leftarrow x_i = n - i; \bar{p_i} \leftarrow x_i = n]$.

Remark 2. In [15], a timed μ-calculus T_μ has been proposed, and MC for T_μ was shown to be PSPACE-hard. T_μ is more expressive than $\mathrm{L}_{\mu,\nu}$ because it allows for fixpoint alternations and it uses a powerful binary operator $\triangleright$ (instead of our modalities $\langle a \rangle$ and $\exists$). In fact the proof of Theorem 1 can be adapted[2] to T_μ and this yields an improved lower bound on the complexity of T_μ MC. Moreover, using techniques from [6], we can prove that the MC problem for T_μ (and the extension of $\mathrm{L}_{\mu,\nu}$ with alternations) is in EXPTIME, and is thus EXPTIME-complete. To the best of our knowledge this is the first precise characterization of the complexity of MC for this logic.

Theorem 2. *The model checking problem for L_ν^- is PSPACE-complete. Moreover the specification complexity of L_ν^- MC is PSPACE-complete. The program complexity of L_ν^- MC is in P, if the integer constants in the automata are represented in unary.*

Proof. **PSPACE membership:** A nondeterministic model checking algorithm in PSPACE can be easily defined by considering the parts of the region graph associated to $A \models \varphi$ only when they are required. The difference with L_ν is that we do not need to compute arbitrary sets of configurations for fixpoints.
PSPACE-hardness: Let $\Phi = Q_1 p_1 \ldots Q_n p_n.\varphi$ be an instance of the QBF (Quantified Boolean Formulae) problem, where each $Q_i \in \{\exists, \forall\}$ and φ is a propositional formula over the p_i's. We reduce the validity of Φ to a model checking problem. Consider the TA $A_{(\varphi)}$ in Figure 1 and the L_ν^- formula

$$\tilde{\Phi} = \exists(\langle a_1 \rangle\, \mathtt{tt} \wedge O_1(\exists(\langle a_2 \rangle\, \mathtt{tt} \wedge O_2 \ldots \exists(\langle a_n \rangle\, \mathtt{tt} \wedge O_n \langle sat \rangle\, \mathtt{tt}))))$$

where O_i is $\langle a_i \rangle$ (resp. $[a_i]$) if Q_i is $\exists$ (resp. $\forall$). Clearly $A_{(\varphi)} \models \tilde{\Phi}$ iff Φ is valid.
Specification complexity: In fact any QBF instance can be encoded as a problem of the form $nil \models \Phi$, with $\Phi \in \mathrm{L}_\nu^-$, by using formula clocks. This entails the PSPACE-hardness of specification complexity.
Program complexity: Let φ be a given L_ν^- formula. We can define a polynomial (in $|A|$) algorithm by building the pertinent part of the region graph in an "on the fly" manner. The key points are that (1) deciding if φ holds for a TA A needs to consider only sequences with at most $|\varphi|$ action transitions and (2) between two action transitions the number of possible delay transitions is bounded by $2(|C_A| + |K|)(\max(\mathtt{MCst}(A), \mathtt{MCst}(\varphi)) + 1)$ which is polynomial in $|A|$ if $\mathtt{MCst}(A)$ is given in unary. The time complexity of such an algorithm is in $O(|A|^{2|\varphi|})$ and, as φ is fixed, the program complexity is in P. $\square$

[2] For ex. by considering a formula like: $\mu X.\mathtt{accept} \vee [\mathtt{tt} \triangleright (p_o \wedge \neg(\mathtt{tt} \triangleright (p_e \wedge \neg X)))]$ where p_e (resp. p_o) marks even (resp. odd) states.

Note that some of our proofs are based upon the realization that the MC problems of the form $nil \models \varphi$ (where φ is a formula in any of the logics considered so far) are just as hard as the MC problems for arbitrary TA. Thus the worst-case complexity of MC for these real-time logics may be seen as deriving solely from the use of clocks in formulae. This pattern will remain true for all the property languages we study in what follows, except $SBLL^-$ and $L^-_{\forall S}$.

The property language L_s has been introduced in [19] as a sub-language of L_ν that allows for more efficient model checking algorithms. To the best of our knowledge, however, such an intention has not been supported yet by precise complexity theoretic considerations. These we now proceed to present.

Theorem 3. *The complexity of L_s MC is PSPACE-complete. Moreover, the specification and program complexities of L_s MC are also PSPACE-complete.*

Proof. **PSPACE membership:** For every L_s formula φ, it is possible to build a TA T_φ such that, for any TA A, $A \models \varphi$ iff a reject node of T_φ is not reachable in the parallel composition $(A|T_\varphi)$ [2]. The size of T_φ is linear in that of φ and $(A|T_\varphi)$ can be seen as a new TA $\bar{A}$ corresponding to the product $A \times T_\varphi$. The reduction of $A \models \varphi$ to a reachability problem for $\bar{A}$ is done in polynomial time, and thus gives the PSPACE membership.
PSPACE-hardness: A reachability question for node n in a TA A can be reduced to checking that $A \not\models \mathtt{max}(X, [\mathtt{in\_n}]\mathtt{ff} \wedge [a]X \wedge \forall\!\!\!\forall X)$ if we suppose that every edge in A has label a, except for a new transition $\langle n, \mathtt{tt}, \mathtt{in\_n}, \emptyset, n\rangle$.
Specification complexity: It is possible to reduce reachability in a linear bounded nondeterministic Turing machine $\mathcal{M}$ with input w to a problem of the form $nil \models \Phi_{\mathcal{M},w}$ by means of the same kind of encoding used for L_ν.
Program complexity: It is PSPACE-complete because the formula expressing the reachability problem does not depend on the input automaton. □

Theorem 4. *The model checking problem for L_s^- is coNP-complete, as is the specification complexity of model checking. The program complexity of L_s^- is in P, if the constants in the input automata are represented in unary.*

The property languages $SBLL$ and $L_{\forall S}$ have the same complexity:

Theorem 5. *The complexity of $SBLL$ and $L_{\forall S}$ model checking is PSPACE-complete. Moreover we have that the specification and program complexities of $SBLL$ and $L_{\forall S}$ MC are also PSPACE-complete.*

For the property languages $SBLL^-$ and $L^-_{\forall S}$, we obtain the following result:

Theorem 6. *The MC problem for $SBLL^-$ and $L^-_{\forall S}$ is PSPACE-complete. The specification complexity of MC for $SBLL^-$ and $L^-_{\forall S}$ is coNP-hard, and is coNP-complete if constants in the formulae are represented in unary. Finally, the program complexity of MC for $SBLL^-$ and $L_{\forall S}$ is PSPACE-complete.*

There is an implicit recursion (over τ and delay transitions) which is hidden in the semantics of the $SBLL^-$ operator $\forall\!\!\!\forall$, and this recursion is sufficient to make $SBLL^-$ and $L^-_{\forall S}$ model checking PSPACE-hard.

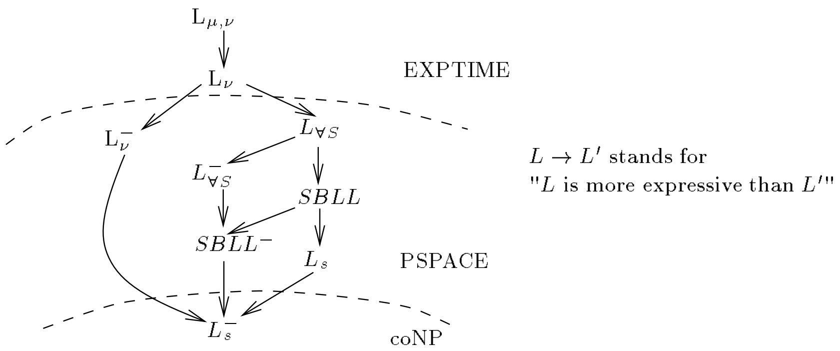

Fig. 2: Expressiveness vs complexity of model checking

Concluding remarks. The relationships between the relative expressive power of the property languages that we have considered, and the complexity of their model checking problems is summarized in Figure 2. (There $L \longrightarrow L'$ means that any model checking problem $A \models \varphi$ with $\varphi \in L'$ can be reduced in linear time to a verification $\tilde{A} \models \tilde{\varphi}$ with $\tilde{\varphi} \in L$.)

Note that, for every specification language we consider, the proof of $\mathcal{C}$-hardness of the MC problem uses formulae without clocks. This implies that the problems $\mathcal{A}_{+t} \stackrel{?}{\models} \Psi$ and $\mathcal{A}_{+t} \stackrel{?}{\models} \Psi_{+t}$ have the same complexity. The remark about the complexity of MC problems of the form $nil \models \varphi$ shows that $\mathcal{A} \stackrel{?}{\models} \Psi_{+t}$ and $\mathcal{A}_{+t} \stackrel{?}{\models} \Psi_{+t}$ also have the same complexity. Therefore the complexity of MC does not depend on whether time is added to the model, to the specification or to both.

References

1. L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen, *The power of reachability testing for timed automata*, in Proc. of FSTTCS'98, LNCS 1530, December 1998, pp. 245–256.
2. L. Aceto, A. Burgueño, and K. G. Larsen, *Model checking via reachability testing for timed automata*, in Proc. of TACAS '98, LNCS 1384, April 1998, pp. 263–280.
3. R. Alur, *Techniques for Automatic Verification of Real-time Systems*, PhD thesis, Stanford University, 1991.
4. R. Alur, C. Courcoubetis, and D. Dill, *Model-checking in dense real-time*, Information and Computation, 104 (1993), pp. 2–34.
5. R. Alur and D. Dill, *A theory of timed automata*, Theoretical Computer Science, 126 (1994), pp. 183–235.
6. O. Bernholtz, M. Y. Vardi, and P. Wolper, *An automata-theoretic approach to branching-time model checking*, in Proc. of the 6th. International Conference on Computer-Aided Verification, CAV'94, D. Dill, ed., vol. 818 of Lecture Notes in Computer Science, California, USA, June 1994, Stanford, Springer-Verlag.
7. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.

8. E. M. Clarke, E. A. Emerson, and A. P. Sistla, *Automatic verification of finite state concurrent system using temporal logic*, ACM Trans. on Programming Languages and Systems, 8 (1986), pp. 244–263.
9. R. Cleaveland, *A linear-time model-checking algorithm for the alternation-free modal μ-calculus*, Formal Methods in Systems Design, 2 (1993), pp. 121–147.
10. C. Courcoubetis and M. Yannakakis, *Minimum and maximum delay problems in real-time systems*, Formal Methods in System Design, (1992), pp. 385–415.
11. S. Demri and P. Schnoebelen, *The complexity of propositional linear temporal logics in simple cases (extended abstract)*, in Proc. 15th Ann. Symp. Theoretical Aspects of Computer Science (STACS'98), LNCS 1373, Paris, France, Feb. 1998, Springer Verlag, 1998, pp. 61–72.
12. S. Dziembowski, M. Jurdziński, and D. Niwiński, *On the expression complexity of the modal μ-calculus model checking.* Unpublished manuscript, November 1996.
13. E. A. Emerson, C. S. Jutla, and A. P. Sistla, *On model-checking for fragments of μ-calculus*, in Proceedings of the Fifth International Conference Computer Aided Verification, Elounda, Greece, July 1993, C. Courcoubetis, ed., vol. 697 of Lecture Notes in Computer Science, Springer-Verlag, 1993, pp. 385–396.
14. M. Hennessy and R. Milner, *Algebraic laws for nondeterminism and concurrency*, J. Assoc. Comput. Mach., 32 (1985), pp. 137–161.
15. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, *Symbolic model checking for real-time systems*, Information and Computation, 111 (1994), pp. 193–244.
16. D. Kozen, *Results on the propositional μ-calculus*, Theoretical Computer Science, 27 (1983), pp. 333–354.
17. F. Laroussinie and K. G. Larsen, *CMC: A tool for compositional model-checking of real-time systems*, in Proc. IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98), Kluwer Academic Publishers, 1998, pp. 439–456.
18. F. Laroussinie, K. G. Larsen, and C. Weise, *From timed automata to logic - and back*, in Proc. of the 20th. International Symposium on Mathematical Foundations of Computer Science, MFCS'95, J. Wiedermann and P. Hájek, eds., vol. 969 of Lecture Notes in Computer Science, Prague, Czech Republic, August 28 - September 1 1995, Springer-Verlag, pp. 529–539.
19. K. G. Larsen, P. Pettersson, and W. Yi, *Model-checking for real-time systems*, in Proceedings of the 10th International Conference on Fundamentals of Computation Theory, H. R. (Ed.), ed., Dresden, Germany, August 1995, LNCS 965, pp. 62–88.
20. ———, *UPPAAL in a Nutshell*, Journal of Software Tools for Technology Transfer, 1 (1997), pp. 134–152.
21. O. Lichtenstein and A. Pnueli, *Checking that finite state concurrent programs satisfy their linear specification*, in Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages, New Orleans, Louisiana, Jan. 1985, pp. 97–107.
22. A. P. Sistla and E. M. Clarke, *The complexity of propositional linear temporal logics*, J. Assoc. Comput. Mach., 32 (1985), pp. 733–749.
23. T.A. Henzinger and P.-H. Ho, and H. Wong-Toi, *HyTech: A Model Checker for Hybrid Systems*, Journal of Software Tools for Technology Transfer, 1 (1997), pp. 110–122.
24. S. Yovine, *Kronos: A Verification Tool for real-Time Systems*, Journal of Software Tools for Technology Transfer, 1 (1997), pp. 123–133.