ELSEVIER

The Journal of Logic and
Algebraic Programming 52–53 (2002) 7–51

THE JOURNAL OF
LOGIC AND
ALGEBRAIC
PROGRAMMING

www.elsevier.com/locate/jlap

Is your model checker on time? On the complexity of model checking for timed modal logics

Luca Aceto[a,1], François Laroussinie[b,*]

[a] *BRICS, Department of Computer Science, Aalborg University, Fredrik Bajers Vej 7-E, DK-9220 Aalborg Ø, Denmark*[2]

[b] *Laboratoire Spécification et Vérification, CNRS UMR 8643, ENS de Cachan, 61 av. du Pr. Wilson, 94235 Cachan, France*

Abstract

This paper studies the structural complexity of model checking for several timed modal logics presented in the literature. More precisely, we consider (variations on) the specification formalisms used in the tools CMC and UPPAAL, and fragments of a timed μ-calculus. For each of the logics, we characterize the computational complexity of model checking, as well as its specification and program complexity, using (parallel compositions of) timed automata as our system model. In particular, we show that the complexity of model checking for a timed μ-calculus interpreted over (networks of) timed automata is EXPTIME-complete, no matter whether the complexity is measured with respect to the size of the specification, of the model or of both. All the flavours of model checking for timed versions of Hennessy–Milner logic, and the restricted fragments of the timed μ-calculus studied in the literature on CMC and UPPAAL, are shown to be PSPACE-complete or EXPTIME-complete. Amongst the complexity results offered in the paper is a theorem to the effect that the model checking problem for the sublanguage L_s of the timed μ-calculus, proposed by Larsen, Pettersson and Yi, is PSPACE-complete. This result is accompanied by an array of statements showing that any extension of L_s has an EXPTIME-complete model checking problem. We also argue that the model checking problem for the timed propositional μ-calculus T_μ is EXPTIME-complete, thus improving upon results by Henzinger, Nicollin, Sifakis and Yovine.

Keywords: Model checking; Timed automata; Timed modal logics; Complexity of verification

1. Introduction

The model checking approach to the computer-aided verification of finite-state programs, first proposed in [25,62], has gained considerable acceptance in the last few years,

[*] Corresponding author. Fax: +33-1-47-40-24-64.
E-mail addresses: luca@cs.auc.dk (L. Aceto), fl@lsv.ens-cachan.fr (F. Laroussinie).
[1] Fax: +45-98-15-98-89.
[2] Basic Research in Computer Science, Centre of the Danish National Rsearch Foundation.

in light of its impressive applications in the design and verification of circuits and distributed protocols (see, e.g., [15,19,28,27] and the references therein). In model checking, we establish the correctness of a system with respect to a given specification by showing that a mathematical model of it satisfies correctness criteria expressed in some formal language. Such a language often takes the form of a temporal logic, like CTL [24], or of a modal logic with fixed points, like the modal μ-calculus [50].

The choice of an appropriate specification language for the application at hand is often the result of a trade-off between the contrasting issues of expressiveness and complexity. On the one hand, the use of a highly expressive logic allows one to describe many interesting behaviours of concurrent systems. On the other, since model checking of realistic systems relies on fully automatic computer support, it is crucial to use a specification language which admits efficient model checking algorithms.

The extension of model checking to the specification and verification of real-time systems has been thoroughly studied in the last few years. This has led to the development of specification logics that extend standard untimed formalisms with the quantitative analysis of timing constraints (see, e.g., [6,8,11,23,40,42,54,65]), and to important theoretical results investigating the limits of decidability for model checking. This theory is now embodied in verification tools like HyTech [70], Kronos [76] and UPPAAL [58], which have been successfully used in the verification of non-trivial systems (see, e.g., [18,45,60]). These applications indicate 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 the real-time modal logics, and variations thereof, used as specification formalisms in the verification tools CMC [53] and UPPAAL.

A complexity-theoretic analysis of the model checking problem yields useful guidelines for selecting a specification logic which offers a good compromise between expressiveness and complexity—see, e.g., [72] and the references therein for a comprehensive complexity-theoretic perspective on model checking for linear- and branching-time logics for reactive systems, which has been one of the main sources of inspiration for this work.

In the remainder of this section, we introduce the different ways of measuring the complexity of model checking problems, outline our main results, and compare them with those for the untimed case. We end the introduction with a discussion of related work.

The 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 [12,14,25,29]. In the timed case, most of the model checking problems considered in literature are PSPACE-hard [5,6,31,42]. 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., [32,37,67]) study in detail the effect of the temporal operators, the number of atomic propositions 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 property languages: L_ν has been introduced in [54], and is the specification language used in the tool CMC [53]; L_s is a fragment of L_ν which has been proposed in [57] in order to improve the efficiency of model checking in practice; SBLL [2] (for Safety and Bounded Liveness Logic) and $L_{\forall S}$ [1] have been introduced for their properties with respect to 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 (networks of) timed automata [7] as our system model. As argued by Lichtenstein and Pnueli [59], 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. (The 'Lichtenstein–Pnueli Thesis' is that, since models of systems are usually large, whereas specifications are usually small, it is the program complexity of model checking that is the most significant measure of its feasibility.) In this paper we offer complexity results for these three different views of the model checking problem for the property languages 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 our study by analyzing the complexity of model checking for a timed μ-calculus, denoted by $L^+_{\mu,\nu}$, and for its alternation-free fragment $L_{\mu,\nu}$ (AFMC) (Section 3.2). 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 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 (Theorem 9). We show, furthermore, that, perhaps surprisingly, the model checking problem for L_ν—and a fortiori for the timed AFMC—is already EXPTIME-hard even if we fix the model to be the inactive process without clocks, nil (see Lemma 13). We feel that this result is more than a theoretical curiosity. Indeed, one of the most successful approaches to model checking for variations on the modal μ-calculus is partial model checking—as presented in, e.g., [13,52]—upon which the verification engine of a tool like CMC is based. In this approach, model checking problems for (networks of timed) automata are reduced to checking properties, expressed in the logic L_ν, of the process nil. Our results show that, in the worst case, such a problem is just as hard as the original one. Similar results apply to the full timed μ-calculus $L^+_{\mu,\nu}$ we study in this paper. We also prove that the model checking problem for the fragment of L_ν without greatest fixed points—essentially, a timed version of Hennessy–Milner logic [41]—is PSPACE-complete (Theorem 11).

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 on

Table 1
Overview of the results (C stands for complete)

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

the tools CMC and UPPAAL. 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 greatest fixed points. 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 (see Theorem 16). 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 [57] to the effect that model checking is easier for L_s than for L_ν. 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 [7].

Since the syntax of the property language L_s contains several seemingly ad hoc restrictions, it is informative to study the structural complexity of model checking for the extensions of this language obtained by removing each of the constraints on the syntax of L_ν imposed in [57]. We prove an array of statements to the effect that the resulting extensions of L_s have an EXPTIME-complete model checking problem (Section 3.4). Such results offer some further justification for the choices made in op. cit.

An overview of the main results we have obtained is presented in Table 1, where L^{-} denotes the fixed point free fragment of a property language L. As argued in Appendix D, the results presented in this paper are robust with respect to changes in the type of guards allowed in timed automata.

Model checking concurrent programs

The aforementioned results on the complexity of model checking are based on the use of timed automata as our model for real-time systems. However, in practice, most real-time systems contain several communicating components, and may be modelled as parallel compositions of timed automata. We denote by *model checking for concurrent programs* the problem of deciding if a modal logic formula holds for a concurrent timed system (viz. a parallel composition of timed automata). Already in the untimed setting, model checking suffers from the so-called state explosion problem in the presence of concurrency. Hence a characterization of the complexity of the model checking problem, for the property languages we study, in the presence of concurrency would yield a more realistic assessment of the hardness of the task of model checking real-time systems. Here we prove that, for all the property languages we consider, unlike in the untimed setting, the concurrency feature does not increase the structural complexity of the model checking problems (Section 4).

Table 2
Timed vs untimed formalisms (C stands for complete)

	$A \models \varphi$		$(A_1 \mid \cdots \mid A_n) \models \varphi$
	Untimed	Timed	(Un)timed
Reachability	NLOGSPACE-C	PSPACE-C	PSPACE-C
(T)CTL	P-C	PSPACE-C	PSPACE-C
AFMC	P-C	EXPTIME-C	EXPTIME-C
full μ-calculus	UP $\cap$ co-UP	EXPTIME-C	EXPTIME-C

An explanation of this phenomenon may be gleaned by comparing our results with those for untimed formalisms.

Comparison with results for untimed formalisms

It is instructive to compare the results we present here 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 [36,51]. The model checking problem for the full μ-calculus is in UP $\cap$ co-UP [47]. Note, however, that the (program) complexity of the (alternation-free) μ-calculus for *concurrent* programs is EXPTIME-complete [51], and this matches exactly the complexity results we offer for $L_{\mu,\nu}$ and $L^{+}_{\mu,\nu}$ model checking. It is also interesting to note that the (program) complexity of CTL model checking and of reachability for concurrent programs is PSPACE-complete [49,51], matching the complexity of model checking for TCTL [6] 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.

An overview of the above comparison of the complexity of model checking for untimed formalisms and their timed counterparts is presented in Table 2.

Related work

There is an extensive literature on the complexity of model checking for linear- and branching-time logics for (concurrent) reactive systems (see, e.g., [32,51,55,64,67,72,73] and the references therein). In particular, [51] offers an automata-theoretic approach to the model checking problem for the branching-time logics CTL, CTL* and the μ-calculus, and gives improved space-complexity bounds for this problem. It will be clear to the readers of this study that we have been greatly inspired by the developments in op. cit.—especially in the technical developments related to establishing lower bounds on the complexity of the model checking problems we consider, and in the study of the model checking problems for concurrent programs. The (program) complexity of LTL model checking for concurrent programs was shown to be PSPACE-complete in the classic study [73], also by means of automata-theoretic techniques.

The literature on the complexity of verification for untimed reactive systems is also rich in results on the complexity of *implementation verification*. In implementation verification, one models both implementations and specifications by means of transition systems. The verification task then consists in checking that the behaviour of the model of an implementation correlates with the behaviour of the model of a specification. The complexity of implementation verification for (concurrent) programs is studied in, e.g., [39,56,63], where both trace- and tree-based approaches are investigated, with and without fairness constraints.

Related literature dealing with the complexity of model checking for real-time logics, and the expressiveness of such languages, has been extensively mentioned throughout this introductory section. Surveys of results in this area of research may be found in, e.g., [8–10,74]. Recent interesting work by Hirshfeld and Rabinovich [43,44] has striven to develop real-time logics which are firmly grounded in the Mathematical Logic tradition, and in the framework of Monadic Logic of Order in particular. The work presented ibidem mostly focuses on expressiveness and decidability issues. For example, [44] introduces Quantitative Temporal Logic (QTL), shows that the satisfiability problem for this logic is PSPACE-complete using techniques from Mathematical Logic, and argues that QTL is as expressive as any of its rivals in the literature. Behavioural relations between timed automata and the relationship with timed logics (via characteristic formulae) are studied in [3].

This paper is an expanded version of [4]. Apart from providing proofs of results that were announced ibidem, the current paper offers several new developments. In particular, we point out that, amongst other things:

- we deal with the full timed μ-calculus $L^{+}_{\mu,\nu}$,
- we study the structural complexity of model checking over networks of timed automata for the real-time logics we consider,
- we analyze the complexity of model checking for extensions of the property language L_s—thus mapping the territory between this property language and L_ν, and
- we study the robustness of our results with respect to changes in the model of timed automata under consideration.

2. Preliminaries

We begin by briefly reviewing a variation on the timed automaton model proposed by Alur and Dill [7] (Section 2.1), and the property languages that will be used in this study (Sections 2.2 and 2.3). The algorithms we shall employ in order to establish upper bounds on the complexity of the model checking problems we consider rely on the region graph construction [7]. For the sake of clarity, this construction and its main properties are outlined in Section 2.4. Finally, we recall the basic problems in complexity theory that will find application in our lower bound arguments (Section 2.6).

2.1. Timed automata

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

Let C be a set of clocks. We use $\mathscr{B}(C)$ 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 $\mathscr{B}_k(C)$ for the restriction of $\mathscr{B}(C)$ to expressions whose integer constants belong to $\{0, \ldots, k\}$. Expressions in $\mathscr{B}(C)$ are interpreted over the collection of time assignments. A *time assignment*, or *valuation*, for C is a function from C to $\mathbb{R}_{\geqslant 0}$. We write $\mathbb{R}^{C}_{\geqslant 0}$ (with typical elements u, v) for the collection of valuations for C. Given $g \in \mathscr{B}(C)$ 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 $d \in \mathbb{R}_{\geqslant 0}$, we use $v + d$ to denote the time assignment which maps each clock $x \in C$ to the value $v(x) + d$. For every set of clocks C' included in C, we write $[C' \to 0]v$ for 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*,
- $n_0 \in N$ is the *initial node*,
- C is a finite set of *clocks*, and
- $E \subseteq N \times \mathscr{B}(C) \times \mathsf{Act} \times 2^C \times N$ is a finite set of *edges*. The quintuple $\langle n, g, a, r, n'\rangle \in E$ 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*). In lieu of $\langle n, g, a, r, n'\rangle$, we shall sometimes use the more suggestive $n \xrightarrow{g,a,r} n'$, with g (resp. r) possibly omitted if $g = \mathtt{tt}$ (resp. $r = \emptyset$).

We use $\mathtt{MCst}(A)$ to denote the largest integer constant occurring in the guards of A.

In the remainder of this study, we shall write *nil* for the one node timed automaton $\langle \mathsf{Act}, \{n_0\}, n_0, \emptyset, \emptyset\rangle$.

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) $\mathscr{T}_A = \langle \mathscr{S}_A, \mathsf{Act} \cup \mathscr{D}, s^0, \longrightarrow\rangle$, where $\mathscr{S}_A$ is the set of states of A, s^0 is the initial state of A, and $\longrightarrow$ is the transition relation defined as follows:

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

The reader will readily realize that the TLTS determined by the timed automaton *nil* consists of one state s^0 of the form $(n_0, -)$ and uncountably many delay transitions of the form $s^0 \xrightarrow{\epsilon(d)} s^0$ with $d \in \mathbb{R}_{\geqslant 0}$.

Definition 2. Let $A = \langle \mathsf{Act}, N, n_0, C, E\rangle$ be a timed automaton. We say that a node $n \in N$ is *reachable* in A iff there is a sequence of transitions in $\mathscr{T}_A$ leading from the initial state s^0 to a state of the form (n, v).

Remark 3. 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 in this paper will still hold for extended TAs. Indeed, given a complexity class $\mathscr{C}$, having a $\mathscr{C}$-hardness result for (simple) TAs implies the same for extended TAs, while having a $\mathscr{C}$ membership result for extended TAs implies the same for TAs. Every $\mathscr{C}$ membership result presented here could be easily adapted to extended TAs. This is because such results are based on the size of the region graph (cf. Section 2.4), that is not bigger for extended TAs than for the TAs introduced in Definition 1 (since adding invariants can only remove some states and delay transitions).

2.2. A timed modal μ-calculus

We now define $L^{+}_{\mu,\nu}$, a timed modal μ-calculus inspired by the logic L_ν [54].

Definition 4. Let K be a finite, non-empty set of clocks (disjoint from C), and Id be a countably infinite set of identifiers (ranged over by X, Y). The set $L^{+}_{\mu,\nu}$ of formulae over K and Id is generated by the following grammar:

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

where $a \in \mathsf{Act}$, $g \in \mathscr{B}(K)$, K' is a subset of 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 a $\mathtt{max}(X, \varphi)$ operator.

We use $\mathtt{MCst}(\varphi)$ to denote the largest integer constant occurring in the clock constraints in φ.

New operators like $\mathtt{tt}$, $\mathtt{ff}$, $g \Rightarrow \psi$ (read "g implies ψ") can be easily defined. For example, $\mathtt{tt}$ and $\mathtt{ff}$ are shorthands for $x \geqslant 0$ and $x < 0$, respectively, for some $x \in K$.

Given a timed automaton A, we interpret formulae in $L^{+}_{\mu,\nu}$ with respect to *extended configurations* (n, v, u), where (n, v) is a configuration of A and u is a time assignment for K. Whereas the classic modal operators $\langle a\rangle$ and $[a]$ deal with action transitions [41], 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 are used as stopwatches for measuring the time elapsing between states of the system. The formula $(K' \ \underline{\mathtt{in}}\ \varphi)$ initializes the formula clocks in K' to 0 in φ. The constraints g are used to compare the values of formula clocks in the current extended configuration with integer values. Finally, an extended configuration satisfies $\mathtt{max}(Y, \varphi)$ (resp. $\mathtt{min}(Y, \varphi)$) if it belongs to the largest (resp. least) solution of the equation $Y = \varphi$ over the complete lattice of sets of extended configurations. The existence of these solutions is guaranteed by standard fixed point theory [71].

To define the formal semantics of $L^{+}_{\mu,\nu}$, it is convenient to consider open formulae—i.e., formulae which may contain occurrences of identifiers which are not bound by least or greatest fixed point operators. Given a timed automaton A, an *environment* is a mapping ρ from identifiers in Id to sets of extended configurations of A. For an environment ρ, identifier X and subset of extended states S, we write $\rho[X \mapsto S]$ for the environment mapping X to S, and acting like ρ on all the other identifiers. The formal semantics of $L^{+}_{\mu,\nu}$ (presented in Table 3) associates with every formula $\varphi \in L^{+}_{\mu,\nu}$ and environment ρ the set of extended states $[\![\varphi]\!]\,\rho$ that satisfy φ, under the assumption that each identifier X is satisfied by the extended states in $\rho(X)$. The interested reader will find more details on this definition in, e.g., [50]. If φ is a closed formula, then the collection of extended states satisfying it is independent of the environment ρ, and will be written $[\![\varphi]\!]$. In the sequel, for every extended state (n, v, u) and closed formula φ, we shall write $(n, v, u) \models \varphi$ (read '(n, v, u) satisfies φ') in lieu of $(n, v, u) \in [\![\varphi]\!]$. For a timed automaton A and closed formula φ, the suggestive shorthand $A \models \varphi$ will be used in lieu of $(n_0, [C \to 0], [K \to 0]) \models \varphi$.

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

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

Table 3
Semantics of $L^{+}_{\mu,\nu}$

$$\begin{aligned}
[\![g]\!]\,\rho &\stackrel{\text{def}}{=} \{(n,v,u) \mid g(u) = \mathtt{tt}\}\\
[\![\varphi_1 \vee \varphi_2]\!]\,\rho &\stackrel{\text{def}}{=} [\![\varphi_1]\!]\,\rho \cup [\![\varphi_2]\!]\,\rho\\
[\![\varphi_1 \wedge \varphi_2]\!]\,\rho &\stackrel{\text{def}}{=} [\![\varphi_1]\!]\,\rho \cap [\![\varphi_2]\!]\,\rho\\
[\![\langle a\rangle\varphi]\!]\,\rho &\stackrel{\text{def}}{=} \{(n,v,u) \mid \exists\,(n',v').\ (n,v)\xrightarrow{a}(n',v') \text{ and } (n',v',u) \in [\![\varphi]\!]\,\rho\}\\
[\![[a]\,\varphi]\!]\,\rho &\stackrel{\text{def}}{=} \{(n,v,u) \mid \forall\,(n',v').\ \ (n,v)\xrightarrow{a}(n',v')\ \ \Rightarrow\ \ (n',v',u) \in [\![\varphi]\!]\,\rho\}\\
[\![\exists\!\!\exists\,\varphi]\!]\,\rho &\stackrel{\text{def}}{=} \{(n,v,u) \mid \exists d \in \mathbb{R}_{\geqslant 0}.\ (n,v+d,u+d) \in [\![\varphi]\!]\,\rho\}\\
[\![\forall\!\!\forall\,\varphi]\!]\,\rho &\stackrel{\text{def}}{=} \{(n,v,u) \mid \forall d \in \mathbb{R}_{\geqslant 0}.\ (n,v+d,u+d) \in [\![\varphi]\!]\,\rho\}\\
[\![K'\ \underline{\mathtt{in}}\,\varphi]\!]\,\rho &\stackrel{\text{def}}{=} \{(n,v,u) \mid (n,v,[K' \to 0]u) \in [\![\varphi]\!]\,\rho\}\\
[\![X]\!]\,\rho &\stackrel{\text{def}}{=} \rho(X)\\
[\![\mathtt{min}(X,\varphi)]\!]\,\rho &\stackrel{\text{def}}{=} \bigcap\{S \mid [\![\varphi]\!]\,\rho[X \mapsto S] \subseteq S\}\\
[\![\mathtt{max}(X,\varphi)]\!]\,\rho &\stackrel{\text{def}}{=} \bigcup\{S \mid S \subseteq [\![\varphi]\!]\,\rho[X \mapsto S]\}
\end{aligned}$$

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. The interested reader is referred to, e.g., [53] for further examples of property specifications in a fragment of $L^{+}_{\mu,\nu}$.

2.3. Fragments of $L^{+}_{\mu,\nu}$

The following fragments of the logic $L^{+}_{\mu,\nu}$ will be of interest in the remainder of this study:

- The alternation-free fragment of $L^{+}_{\mu,\nu}$ (denoted by $L_{\mu,\nu}$) consists of all the formulae where each occurrence of an identifier X, which is bound by a $\mathtt{min}(X,\varphi)$ (resp. $\mathtt{max}(X,\varphi)$) operator, cannot occur in a subformula of φ of the form $\mathtt{max}(Y,\psi)$ (resp. $\mathtt{min}(Y,\psi)$).
- The logic L_ν [54] is the fragment of $L_{\mu,\nu}$ in which only the use of greatest fixed points is allowed.
- The property language L_s [57] 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 $g \in \mathscr{B}(K)$) is allowed.
- The property languages $SBLL$ and $L_{\forall S}$ extend L_s, and are interpreted over a slightly different kind of timed automata where
  (1) $\mathscr{U}$ is a subset of Act such that any edge labeled with $a \in \mathscr{U}$ has the guard $\mathtt{tt}$, and
  (2) Act contains the label τ used to denote an internal action of automata.
  Moreover, the semantics of these property languages is based on a different notion of satisfaction relation (denoted by $\vdash$ in what follows) compared with $L^{+}_{\mu,\nu}$, L_ν or L_s. A formula φ holds for an extended configuration (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') such that $(n,v)\xrightarrow{\tau}{}^{*}(n',v')$, we have that $(n',v')\xrightarrow{a}(n'',v'')$ implies $(n'',v'',u) \vdash \varphi$; and
  - $(n,v,u) \vdash \forall\!\!\forall\varphi$ iff for every (n',v') reached from (n,v) by using τ- 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 $a \in \mathscr{U}$. $L_{\forall S}$ extends $SBLL$ with new operators $\forall\!\!\forall_S$ with $S \subseteq \mathscr{U}$. 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- and delay transitions (with total duration d), but delay transitions with a positive duration 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 $\overline{[a]\psi} = \mathtt{max}(X, [a]\overline{\psi} \wedge [\tau]X)$ (see Appendix A for a complete translation). An important property [1,2] of $SBLL$ and $L_{\forall S}$ is that their model checking problem can be reduced to a reachability problem as follows: 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 synchronized parallel composition $(A|T_\varphi)$. Moreover it has been shown that $L_{\forall S}$ is expressive enough to encode any reachability property over the kind of timed automata used in the tool UPPAAL [1]. The interested reader is referred to op. cit. for more information on $SBLL$ and $L_{\forall S}$.

For each property language L, we shall use L^- to denote its fixed point free fragment.

Remark 5. In [42], a timed μ-calculus T_μ has been proposed, and the model checking problem for T_μ was shown to be PSPACE-hard. The logic T_μ uses a powerful binary operator $\rhd$ (instead of our modalities $\langle a \rangle$ and $\exists\!\!\exists$). Our subsequent EXPTIME-completeness result for the model checking problem for the logic $L^+_{\mu,\nu}$ (see Theorem 9) can be easily adapted to T_μ, yielding an improved lower bound on the complexity of T_μ model checking (see Remark 10).

2.4. The region graph construction

For the sake of clarity, we now briefly review some of the basic techniques and results from the algorithmics of timed automata that will be needed in the remainder of this study. Our presentation will be necessarily sketchy, and we heartily refer the reader to the references for full details.

Regions of time assignments

Automatic verification of timed systems modelled as (networks of) timed automata is possible despite the uncountably infinite number of configurations associated with a timed automaton. The decision procedure for the problem $A \models \varphi$ is based on the well-known *region technique* [6]. Let C^+ denote the set of clocks $C \cup K$. Given A and φ, it is possible to partition the uncountably infinite set of time assignments over C^+ into a finite number of *regions*, in such a way that two extended configurations (n, u) and (n, v), where $u, v \in \mathbb{R}_{\geqslant 0}^{C^+}$ are in the same region, satisfy the same subformulae of φ. Formally, the regions can be defined as the equivalence classes induced by the equivalence relation over valuations defined thus: two valuations u and v are in the same region iff they satisfy the same clock constraints in $\mathscr{B}_M(C^+)$, where $M = \max(\mathtt{MCst}(A), \mathtt{MCst}(\varphi))$.

We write $[u]$ for the region which contains the time assignment u, and use $\mathscr{R}_k^{Cl}$ (ranged over by γ) to denote the (finite) set of all regions for a set Cl of clocks and an integer constant k. Given a region $[u]$ in $\mathscr{R}_k^{Cl}$ and $C' \subseteq Cl$, we define the reset operator thus: $[C' \to 0][u] = [[C' \to 0]u]$. Moreover, given a region γ, its (unique) successor region in $\mathscr{R}_k^{Cl}$, denoted by $succ(\gamma)$, is the region γ' such that:

- $\gamma' = \gamma$ if $\gamma(x > k) = \mathtt{tt}$ for every $x \in Cl$,
- otherwise $\gamma' \neq \gamma$, and for every $u \in \gamma$ there exists $d \in \mathbb{R}_{\geqslant 0}$ with
  - $[u + d] = \gamma'$, and
  - $[u + d'] \in \{\gamma, \gamma'\}$ for every $d' < d$.

We write $\gamma' \in succ^*(\gamma)$ if $\gamma' = succ^i(\gamma)$ for some non-negative integer i.

The region graph

Given a timed automaton $A = \langle \mathsf{Act}, N, n_0, C, E\rangle$, a set K of formula clocks (disjoint from C) and an integer constant M, with $M \geqslant \mathtt{MCst}(A)$, we can define a symbolic semantics [54] for A over the finite transition system $(S, \rightarrow)$, called the *region graph*, defined thus:

- the set of symbolic states S is $N \times \mathscr{R}_M^{C^+}$, and
- $\rightarrow = (\bigcup_{a\in\mathsf{Act}} \xrightarrow{a}) \cup \xrightarrow{succ}$, where, for every symbolic state (n, γ),

$$(n,\gamma) \xrightarrow{a} (n',\gamma') \text{ iff } \exists\langle n, g, a, r, n'\rangle \in E : g(\gamma) = \mathtt{tt} \text{ and } \gamma' = [r \rightarrow 0]\gamma,$$
$$(n,\gamma) \xrightarrow{succ} (n',\gamma') \text{ iff } n = n' \text{ and } \gamma' = succ(\gamma).$$

Formulae in $L^+_{\mu,\nu}$ can be readily interpreted over the states of the region graph by evaluating the delay modalities with respect to $succ$-transitions, and guards with respect to regions instead of single time valuations. (Cf., e.g., [54] for details. Note that the developments in op. cit. only deal explicitly with L_ν; however, they apply equally well to the whole of $L^+_{\mu,\nu}$.) The resulting 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, region $\gamma \in \mathscr{R}_M^{C^+}$ and valuation $u \in \gamma$, we have that $(n,\gamma) \models \varphi$ iff $(n,u) \models \varphi$. Therefore to decide if $A \models \varphi$ holds, it is sufficient to apply a standard model checking algorithm over the (finite) region graph built from A and φ. (For the sake of clarity, we remark that the approach described above yields a possibly different instance of an untimed model checking problem for every timed automaton A and formula φ.)

In our complexity analyses to follow, we shall need to estimate the size of the region graph. Note that the size of $\mathscr{R}_M^{C^+}$ is in $O(|C^+|! \cdot M^{|C^+|})$. We can estimate the sizes of the set of states S and of the transition relation $\rightarrow$ of the region graph thus:

- $|S|$ is in $O(|N| \cdot |C^+|! \cdot M^{|C^+|})$ and
- $|\rightarrow| = |\bigcup_{a\in\mathsf{Act}} \xrightarrow{a}| + |\xrightarrow{succ}|$ is in $O((|E| + |N|) \cdot |C^+|! \cdot M^{|C^+|})$.

Moreover, for every region γ,

$$\left|\{\gamma' \mid \gamma' = succ^i(\gamma) \text{ for some } i \in \mathbb{N}\}\right| \leqslant 2 \cdot |C^+| \cdot (M + 1).$$

The interested reader is invited to consult, e.g., [6] for details.

2.5. Algorithms over regions

In order to define algorithms operating on region graphs, we shall need some basic operations over regions. We can represent a region γ over a set of clocks C by means of a Difference Bounded Matrix [17,33,75] M_γ, which is a $(|C| + 1) \times (|C| + 1)$ integer matrix s.t. $M_\gamma(i, j) = k$ iff k is the smallest integer s.t. $x_i - y_j \leqslant k$ (with the convention that $x_0 = 0$). (For simplicity we just mention non-strict comparisons. Strict comparisons can also be handled in a similar way.) Usually, DBMs encode convex sets of regions. These data structures are employed in many verification tools for timed systems, and many algorithms over DBMs have been defined. Here we are interested in the following problems:

- building a DBM M_γ from the representation of γ as a set of clock constraints (see, e.g., [7]),
- deciding whether a DBM M corresponds to a region,
- deciding whether a given clock constraint g is satisfied by a region γ,
- deciding (by using their DBM representations) whether a region γ' is reachable from γ by using only *succ*-transitions, and
- computing the DBM $M_{[C' \to 0]\gamma}$ from M_γ.

All these operations can be performed in polynomial time by using standard algorithms [33]. By way of example, we limit ourselves to presenting an algorithm for deciding if $\gamma' \in succ^*(\gamma)$ holds for two regions γ and γ'. Given M_γ, we first compute $M_{\overrightarrow{\gamma}}$, where $\overrightarrow{\gamma}$ denotes the (convex) set of regions reachable from γ by using *succ*-transitions. To this end, it is sufficient to remove the upper bounds on the values of the clocks in the constraints for γ (this is done by using a large enough integer value m_∞ for $M_{\overrightarrow{\gamma}}(i, 0)$). Finally it remains to check if each constraint of γ' implies the corresponding one in $\overrightarrow{\gamma}$. Therefore deciding whether $(n, \gamma) \xrightarrow{succ}{}^{*} (n', \gamma')$ can be done in polynomial time by verifying that:

(1) $n' = n$,
(2) γ' is a region, and
(3) $\gamma' \in succ^*(\gamma)$.

Deciding whether $(n, \gamma) \xrightarrow{a} (n', \gamma')$ holds can also be done in polynomial time: we need to verify if (1) there is an edge $n \xrightarrow{g,a,r} n'$ whose guard g holds for γ, and (2) γ' is the region $[r \to 0]\gamma$.

2.6. Basic problems for complexity analysis

To determine lower bounds on the complexity of model checking problems, we shall reduce several classic problems, whose complexity is well known, to instances of model checking problems. In the remainder of this study, we shall consider reachability in Linear Bounded Turing Machines and validity of Quantified Boolean Formulae for PSPACE-hardness results, reachability in Linear Bounded Alternating Turing Machines for EXPTIME-hardness results, and validity of propositional formulae for coNP-hardness arguments. These we now proceed to sketch briefly, for the sake of completeness. The interested reader is warmly referred to [61] for an encyclopaedic treatment.

Linear Bounded and Alternating Turing Machines

A non-deterministic Turing machine is a quintuple $\mathscr{M} = \langle Q, \Sigma, q_0, q_F, T\rangle$ where

- Q is a finite set of states,
- Σ is a finite alphabet,
- $q_0 \in Q$ is the initial state,
- $q_F \in Q$ is the final state, and
- $T \subseteq Q \times \Sigma \times \Sigma \times \{L, R\} \times Q$ is the set of transitions.

A configuration of $\mathscr{M}$ is a triple $(q, w, i) \in Q \times \Sigma^* \times \mathbb{N}$ where q denotes the current control state, w represents the contents of the tape (with the understanding that cell j contains the symbol $w(j)$, viz. the jth symbol in w, if $0 < j \leqslant |w|$, and the blank symbol otherwise) and i denotes the current position of the tape head. Let $(q, \alpha, \alpha', \delta, q')$ be a transition in T. We say that such a transition is enabled in the configuration (q, w, i) iff $w(i) = \alpha$. In that case, performing it leads to the successor configuration (q', w', i') with $w'(j) = w(j)$ for every $j \neq i$, $w'(i) = \alpha'$, and $i' = i + 1$ (resp. $i' = i - 1$) if $\delta = R$ (resp. $\delta = L$ and $i > 1$). An input word w is accepted by $\mathscr{M}$ iff there is an execution starting from $(q_0, w, 1)$ (the initial configuration of $\mathscr{M}$ on input w) which leads to a configuration whose control state is q_F.

A *Linear Bounded* Turing Machine (LBTM) $\mathscr{M}$ is a non-deterministic Turing Machine which can only use $|w|$ cells of the tape to decide whether an input w is accepted by $\mathscr{M}$ or not. Deciding whether a word is accepted by a LBTM is PSPACE-complete [48].

An *Alternating Turing Machine* (ATM) $\mathscr{M}$ is a non-deterministic Turing machine whose set of states is partitioned into two sets Q_{and} and Q_{or}. We assume without loss of generality that $q_0, q_F \in Q_{\text{or}}$. Let (q, w, i) be a configuration of $\mathscr{M}$. A configuration (q, w, i) with $q \in Q_{\text{or}}$ is said to be *accepting* if either q is the accepting state q_F or there exists a successor configuration (q', w', i') such that (q', w', i') is an accepting configuration. A configuration (q, w, i) with $q \in Q_{\text{and}}$ is said to be *accepting* if it has at least one successor configuration, and each of its successor configurations (q', w', i') is an accepting configuration. A word w is said to be *accepted* by $\mathscr{M}$ iff $(q_0, w, 1)$ is an accepting configuration.

A *Linear Bounded* Alternating Turing Machine (LBATM) is an ATM whose tape head cannot go beyond the end of the input markers. Deciding whether a word is accepted by a LBATM is EXPTIME-complete [22].

Problems from logic

We shall also use the validity problem for Quantified Boolean Formulae (QBF), which is known to be PSPACE-complete [68,69]. An instance of QBF is a formula of the form $\Phi = Q_1 p_1 \cdots Q_n p_n \cdot \varphi$, where φ is a boolean formula (using $\wedge$ and $\vee$) over the set of atomic propositions $P = \{p_1, \ldots, p_n\}$ and their negations $\{\bar{p_1}, \ldots, \bar{p_n}\}$. The quantifier Q_i belongs to $\{\exists, \forall\}$ for every $i \in \{1, \ldots, n\}$. We shall have some use for the following lemma from [32]:

Lemma 6. *Let $\Phi = Q_1 p_1 \cdots Q_n p_n \cdot \varphi$ be an instance of QBF. Then Φ is valid iff there exists a non-empty set $\mathscr{V} \subseteq \{\mathtt{tt}, \mathtt{ff}\}^P$ of boolean valuations such that*:

- **correctness:** *$v \models \varphi$ for every $v \in \mathscr{V}$, and*
- **closure:** *for all $v \in \mathscr{V}$, for all i such that $Q_i = \forall$, there is a valuation $v' \in \mathscr{V}$ such that $v'(p_i) = \neg v(p_i)$ and $v'(p_j) = v(p_j)$ for every $j < i$.*

Finally we shall use the *validity* problem for propositional formulae (viz. "Does a given propositional formula hold for every interpretation of its atomic variables?") which is a well-known coNP-complete problem.

3. Complexity results for model checking

In order to formulate and prove the complexity results to follow, we need to define what is the size of a timed automaton $A = \langle \mathsf{Act}, N, n_0, C, E \rangle$ and of a formula $\varphi \in L^{+}_{\mu,\nu}$. The size $|\varphi|$ of a formula is its length in symbols. (Note that, since each $g \in \mathscr{B}(C)$ is a formula, this also defines the size of a guard g.) We define the size $|A|$ of a timed automaton as $|N| + |C| + \Sigma_{e\in E}|g_e|$. Considering constants represented in unary or binary does not change our results except when it is explicitly mentioned.

3.1. Reachability in timed automata

Before embarking in our analysis of the complexity of model checking for $L^{+}_{\mu,\nu}$ and its sublanguages, we recall a well-known, and important, result [5,31] on the complexity of reachability in timed automata. The proof of such a statement is based upon an encoding of the workings of a Linear Bounded Turing Machine on a given input string by means of a timed automaton which will be used repeatedly in the technical developments to follow.

Lemma 7. *Reachability in timed automata is PSPACE-complete.*

Proof. We establish membership in PSPACE and hardness for PSPACE separately.

PSPACE membership. This will follow from the PSPACE membership of the reachability problem for networks of timed automata (see Theorem 31).

PSPACE-hardness. We show that the acceptance problem for LBTMs can be reduced, in polynomial time, to the reachability problem for timed automata. Let $\mathscr{M} = \langle Q, \Sigma, q_0, q_F, T\rangle$ be a non-deterministic Linear Bounded Turing Machine. We assume, without loss of generality, that $\Sigma = \{a, b\}$. Let w_0 be an input word over Σ^*. From $\mathscr{M}$ and w_0 we are going to build, in polynomial time, a timed automaton $A_{\mathscr{M},w_0}$ such that w_0 is accepted by $\mathscr{M}$ iff a distinguished *end* node is reachable in $A_{\mathscr{M},w_0}$.

Let $n = |w_0|$. The timed automaton $A_{\mathscr{M},w_0}$ is constructed as follows. The set of nodes of $A_{\mathscr{M},w_0}$ is $\{(q, i) \mid q \in Q \text{ and } 1 \leqslant i \leqslant n\} \cup \{init, end\}$. Intuitively, a node (q, i) denotes the current state q of $\mathscr{M}$ and the current position i of the tape head. To encode the configurations of $\mathscr{M}$, it is necessary to represent the tape contents. This is done by means of appropriate clock valuations. For each tape cell C_j $(1 \leqslant j \leqslant n)$, we have two clocks x_j and y_j whose values encode the content of C_j as follows: if C_j contains an a (resp. b), we have $x_j = y_j$ (resp. $x_j < y_j$). Writing an a (resp. a b) on cell j will consist in resetting x_j and y_j at the same time (resp. delaying a positive amount of time to ensure that the value of y_j is positive, and thereafter resetting x_j). Note that this encoding of the tape contents is preserved by delay transitions.

The action set of $A_{\mathscr{M},w_0}$ is $\{s_0, s, \mathtt{accept}\}$. For every transition $(q, \alpha, \alpha', \delta, q')$ of machine $\mathscr{M}$, we add, for every possible position of the tape head $i \in \{1, \ldots, n\}$, transitions $(q, i) \xrightarrow{g,s,r} (q', i')$ such that
(1) g is $x_i = y_i$ (resp. $x_i < y_i$) if $\alpha = a$ (resp. $\alpha = b$),
(2) $r = \{x_i, y_i\}$ (resp. $r = \{x_i\}$) if $\alpha' = a$ (resp. $\alpha' = b$), and
(3) $i' = i + 1$ (resp. $i' = i - 1$) if δ is R and $i < n$ (resp. δ is L and $i > 1$).
Finally, we need to add a constraint to ensure time elapsing because, to be sure that resetting x_j will encode the writing of a b in cell C_j, the value of the clock y_j has to be greater than 0. To this end, we use an extra clock t which is reset by every transition, and we add the constraint $t = 1$ to the guard g of the previously described transitions.

The initialization of the tape with the input word w_0 can be encoded by adding a transition $init \xrightarrow{t=1,s_0,r_{w_0}} (q_0, 1)$, where $r_{w_0} = \{x_i \mid w_0(i) = b\} \cup \{t\}$. To detect the accepting run, we add transitions $(q_F, i) \xrightarrow{\mathtt{accept}} end$ for every i. Clearly the node *end* is reachable in $A_{\mathscr{M},w_0}$ iff $\mathscr{M}$ accepts w_0.

Since $A_{\mathscr{M},w_0}$ can obviously be built in polynomial time from $\mathscr{M}$ and w_0, we have thus proven that the acceptance problem for LBTMs can be reduced in polynomial time to the reachability problem in timed automata, which was to be shown. □

Remark 8. The encoding used in the above proof employs comparisons between the values of the clocks x_j, y_j to represent the content of cell C_j. The encoding of an LBTM on input w by means of a timed automaton would be possible by using only comparisons between clocks and integer values in the guards. However, in this case, one step of the LBTM would be simulated by several transitions in the timed automaton encoding it. Nevertheless all the complexity results to follow would still hold. We refer the interested reader to Appendix D for the details of such an alternative encoding.

We now consider the complexity of model checking for the property languages introduced previously.

3.2. The logics L_ν, $L_{\mu,\nu}$ and $L^+_{\mu,\nu}$

We begin by studying the complexity of model checking for the logics L_ν, $L_{\mu,\nu}$ and $L^+_{\mu,\nu}$.

Theorem 9. *The model checking problem for $L^+_{\mu,\nu}$, $L_{\mu,\nu}$ and L_ν is EXPTIME-complete. Moreover, we have that the specification and program complexities of $L^+_{\mu,\nu}$, $L_{\mu,\nu}$ and L_ν model checking are also EXPTIME-complete.*

Proof. EXPTIME membership. This is a consequence of the EXPTIME membership of $L^+_{\mu,\nu}$ model checking for networks of timed automata (see Theorem 32 to follow).

EXPTIME-hardness. We offer a polynomial time reduction from the acceptance of an input w_0 by a LBATM $\mathscr{M}$ to a model checking problem $A_{\mathscr{M},w_0} \models \Phi$, where $A_{\mathscr{M},w_0}$ is the timed automaton used in the proof of Lemma 7, and Φ is an L_ν formula used to encode the behaviour of $\mathscr{M}$ on input w_0. We assume, without loss of generality, that we have a strict alternation of "or" and "and" states in $\mathscr{M}$; moreover we remind the reader that we assume that the initial and final states of $\mathscr{M}$ are "or" states.

By following the same approach proposed in [51] for untimed concurrent systems, the alternating behaviour of $\mathscr{M}$ can be handled by an L_ν formula of the form

$$\Phi = \forall\!\!\forall[s_0]\mathtt{max}\big(X, [\mathtt{accept}]\mathtt{ff} \wedge \forall\!\!\forall\,[s]\,\exists\!\!\exists\,\langle s\rangle X\big). \tag{1}$$

Intuitively Φ is satisfied by $A_{\mathscr{M},w_0}$ if, after the initialization step, the current "or" state is not an accepting state (as required by the subformula $[\mathtt{accept}]\mathtt{ff}$), 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 that $A_{\mathscr{M},w_0} \models \Phi$ iff the LBATM $\mathscr{M}$ does not accept w_0. Indeed Φ is the negation of a formula that uses least fixed points to express that an accepting state is reachable after some sequence of transitions, namely

$$\exists\!\!\exists\langle s_0\rangle\mathtt{min}\big(X, \langle\mathtt{accept}\rangle\mathtt{tt} \vee \exists\!\!\exists\,\langle s\rangle\,\forall\!\!\forall\,[s]X\big).$$

Its negation Φ uses only a greatest fixed point, and can be expressed with L_ν. The use of quantifiers and modalities in the formula allows us to express the alternating form of $\mathscr{M}$'s

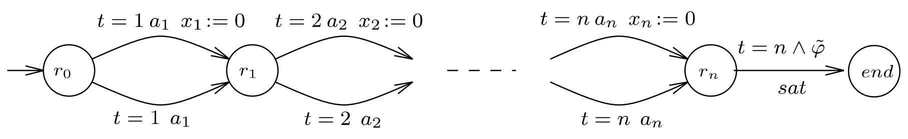
$t = 1\, a_1 \;\; x_1 := 0$
$t = 2\, a_2 \;\; x_2 := 0$
$t = n\, a_n \;\; x_n := 0$
$t = n \wedge \tilde{\varphi}$
r_0
r_1
r_n
end
sat
$t = 1 \;\; a_1$
$t = 2 \;\; a_2$
$t = n \;\; a_n$

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

acceptance conditions. This yields the EXPTIME-hardness of the model checking problem for L_ν, $L_{\mu,\nu}$ and $L^{+}_{\mu,\nu}$.

Program complexity. The claim follows from the proof of EXPTIME-hardness for L_ν model checking, where the formula Φ we have constructed is fixed, and does not depend on the LBATM $\mathscr{M}$.

Specification complexity. The claim is a consequence of Lemma 13 to follow, which states that the model checking problem for the *nil* process is EXPTIME-complete. □

Remark 10. The proof of Theorem 9 can be adapted[3] to the propositional timed μ-calculus T_μ proposed in [42], thus yielding an improved lower bound on the complexity of T_μ model checking. Moreover, the proof used to show EXPTIME membership of $L^{+}_{\mu,\nu}$ model checking over networks of timed automata (see Theorem 32 to follow) can be used *mutatis mutandis* for T_μ. Therefore model checking for T_μ over (networks of) timed automata is EXPTIME-complete. To the best of our knowledge, this is the first precise characterization of the structural complexity of model checking for this logic.

We now proceed to study the computational complexity of model checking for the logic L^{-}_{ν}—that is, the fixed point free fragment of L_ν (and thus of $L_{\mu,\nu}$ and $L^{+}_{\mu,\nu}$). We have:

Theorem 11.
(1) *The model checking problem for L^{-}_{ν} is PSPACE-complete.*
(2) *The specification complexity of L^{-}_{ν} model checking is PSPACE-complete.*
(3) *The program complexity of L^{-}_{ν} model checking is in P, if the integer constants in the timed automata are represented in unary.*

Proof. We prove each statement in turn.

(1) We separately establish membership in, and hardness for, PSPACE.

PSPACE membership. This will follow from the more general argument in Theorem 33.

PSPACE-hardness. Let $\Phi = Q_1 p_1 \cdots Q_n p_n \cdot \varphi$ be an instance of the QBF problem. We offer a polynomial time reduction from the validity of Φ to a model checking problem for L^{-}_{ν}. Consider the timed automaton $A_{(\varphi)}$ and the formula $\tilde{\varphi}$ in Fig. 1, and the following L^{-}_{ν} formula:

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

where O_i is $\langle a_i\rangle$ (resp. $[a_i]$) if Q_i is $\exists$ (resp. $\forall$).

[3] For example by considering the formula $\mu X.\mathtt{accept} \vee [\mathtt{tt} \rhd (p_o \wedge \neg(\mathtt{tt} \rhd (p_e \wedge \neg X)))]$, where p_e (resp. p_o) marks even (resp. odd) states.

Consider a path

$$\begin{aligned}\rho = (r_0, u_0^\rho) &\xrightarrow{\epsilon(1)} (r_0, u_0^\rho + 1) \xrightarrow{a_1} (r_1, u_1^\rho) \cdots \\ &(r_{n-1}, u_{n-1}^\rho) \xrightarrow{\epsilon(1)} (r_{n-1}, u_{n-1}^\rho + 1) \xrightarrow{a_n} (r_n, u_n^\rho)\end{aligned} \tag{2}$$

in the TLTS $\mathscr{T}_{A_{(\varphi)}}$, where u_0^ρ is the valuation setting all clocks to 0. Along such a path,
(1) each action a_i is performed at time i,
(2) each clock x_i is either reset at time i or is never reset, and
(3) the clock t is never reset.
Such a path describes a valuation v_ρ for the atomic propositions $p_1, \ldots, p_n$ according to the following convention:
- if x_i has been reset at time i (i.e., if $u_n^\rho(x_i) = n - i$), then $v_\rho(p_i) = \mathtt{tt}$,
- otherwise (i.e., if $u_n^\rho(x_i) = n$) $v_\rho(p_i) = \mathtt{ff}$.

This entails that at time n (i.e., when the value of t is n), the sat-labelled transition is enabled from the configuration (r_n, u_n^ρ) (i.e., $u_n^\rho \models \tilde{\varphi}$) iff $v_\rho \models \varphi$. We claim that:

$$\Phi \text{ is valid} \quad \Leftrightarrow \quad A_{(\varphi)} \models \tilde{\Phi}. \tag{3}$$

To prove that this does hold, define

$$\widetilde{\Phi_i} \overset{\text{def}}{=} \exists\big(\langle a_{i+1}\rangle \mathtt{tt} \wedge O_{i+1}\widetilde{\Phi_{i+1}}\big) \quad (i = 0, \ldots, n-1)$$

and let $\widetilde{\Phi_n}$ be $\langle sat\rangle\mathtt{tt}$. Note that $\tilde{\Phi} = \widetilde{\Phi_0}$. We establish the two implications of statement (3) separately.

- ($\Rightarrow$) Assume that Φ is valid. This implies that there exists a set of boolean valuations $\mathscr{V}$ satisfying the requirements in Lemma 6. To establish that $A_{(\varphi)} \models \tilde{\Phi}$ holds, it is sufficient to use the following auxiliary lemma, whose proof is an easy induction over $n - i$.

  Lemma 12. *Let Φ be valid and let v be a boolean valuation in a set of boolean valuations $\mathscr{V}$ that satisfies the requirements of Lemma* 6. *Let u_i^v (with $i \leqslant n$) be a timed valuation for the set of clocks $\{x_1, \ldots, x_n, t\}$ such that*:
  - $u_i^v(t) = i$,
  - $u_i^v(x_j) = i$ *if* $j > i$, *and*
  - $u_i^v(x_j) = i$ (*resp.* $u_i^v(x_j) = i - j$) *if* $j \leqslant i$ *and* $v(p_j) = \mathtt{ff}$ (*resp.* $v(p_j) = \mathtt{tt}$).

  Then we have that $(r_i, u_i^v) \models \widetilde{\Phi_i}$.

- ($\Leftarrow$) Assume $A_{(\varphi)} \models \tilde{\Phi}$. We shall show that Φ is valid. To this end, it is sufficient to exhibit a collection $\mathscr{V}$ of boolean valuations satisfying the conditions of Lemma 6.
  Let ρ be a path in $\mathscr{T}_{A_{(\varphi)}}$ of the form (2). We say that ρ is *$\tilde{\Phi}$-consistent* if $(r_i, u_i^\rho) \models \widetilde{\Phi_i}$, for every $i \in \{0, \ldots, n\}$. In particular, if ρ is $\tilde{\Phi}$-consistent, then $u_n^\rho \models \tilde{\varphi}$, and thus $v_\rho \models \varphi$. Note, furthermore, that, if ρ is $\tilde{\Phi}$-consistent and $Q_{i+1} = \forall$ for some $0 \leqslant i < n$, then the path ρ' defined thus

$$\begin{aligned}(r_0, u_0^\rho) &\xrightarrow{\epsilon(1)} (r_0, u_0^\rho + 1) \xrightarrow{a_1} (r_1, u_1^\rho) \cdots (r_i, u_i^\rho) \xrightarrow{\epsilon(1)} (r_i, u_i^\rho + 1) \xrightarrow{a_{i+1}} \\ &(r_{i+1}, u_{i+1}^{\rho'}) \cdots (r_{n-1}, u_{n-1}^{\rho'}) \xrightarrow{\epsilon(1)} (r_{n-1}, u_{n-1}^{\rho'} + 1) \xrightarrow{a_n} (r_n, u_n^{\rho'}),\end{aligned}$$

  is also $\tilde{\Phi}$-consistent, where, for every $j > i$ and clock y, the time valuation $u_j^{\rho'}$ is such that
  - $u_j^{\rho'}(y) = j - (i+1)$, if $y = x_{i+1}$ and $u_{i+1}^\rho(x_{i+1}) = i + 1$ (i.e., if clock x_{i+1} was not reset by the $(i+1)$th action transition in ρ),

- $u_j^{\rho'}(y) = j$, if $y = x_{i+1}$ and $u_{i+1}^{\rho}(x_{i+1}) = 0$ (i.e., if clock x_{i+1} was reset by the $(i+1)$th action transition in ρ), and
- $u_j^{\rho'}(y) = u_j^{\rho}(y)$, otherwise.

Since ρ' is $\tilde{\Phi}$-consistent, it is easy to see that $v_{\rho'} \models \varphi$. It follows that the collection of boolean valuations associated with the set of $\tilde{\Phi}$-consistent paths of $\mathscr{T}_{A_{(\varphi)}}$ is non-empty (since $A_{(\varphi)} \models \tilde{\Phi}$ and $\tilde{\Phi}$ requires the existence of at least one path of the form (2)), and satisfies the correctness and closure requirements from Lemma 6. We may therefore conclude that Φ is valid.

This completes the proof of (3).

(2) **Specification complexity.** Any QBF instance can be encoded as a model checking problem of the form $nil \models \Phi$, with $\Phi \in L_\nu^-$, by using formula clocks (see Lemma 13 to follow). This entails the PSPACE-hardness of the specification complexity of model checking for L_ν^-.

(3) **Program complexity.** Let φ be a given L_ν^- formula. We can define an algorithm, whose worst-case complexity is polynomial in $|A|$, by building the pertinent part of the region graph in an "on the fly" manner. The key points are that

(a) to decide if φ holds for a timed automaton A we only need to consider sequences with at most $|\varphi|$ action transitions, and

(b) between two action transitions the number of possible delay transitions in the region graph is bounded from above by

$$2(|C_A| + |K|)(\max(\texttt{MCst}(A), \texttt{MCst}(\varphi)) + 1),$$

which is polynomial in $|A|$ if $\texttt{MCst}(A)$ is given in unary.

The algorithm to decide if φ holds for a symbolic state (r, γ) proceeds by structural induction over the formula φ:

- To decide if $(r, \gamma) \models \varphi$ holds, with φ of the form $\psi_1 \wedge \psi_2$ or $\psi_1 \vee \psi_2$, we need (in the worst case) to decide whether $(r, \gamma) \models \psi_i$ holds with $i = 1, 2$.
- To decide if $(r, \gamma) \models \varphi$ holds, with φ of the form $\langle a\rangle\psi$ or $[a]\psi$, we need (in the worst case) to decide whether $(r_i, \gamma_i) \models \psi$ holds for every (r_i, γ_i) such that $(r, \gamma) \xrightarrow{a} (r_i, \gamma_i)$. The number of such (r_i, γ_i) is bounded from above by $|A|$.
- To decide if $(r, \gamma) \models \varphi$ holds, with φ of the form $\exists\!\!\exists\psi$ or $\forall\!\!\forall\psi$, we need (in the worst case) to decide whether $(r, succ^i(\gamma)) \models \psi$ holds for every i no larger than $2(|C| + |K_\varphi|)(\max(\texttt{MCst}(A), \texttt{MCst}(\varphi)) + 1)$. Since we assume that $\texttt{MCst}(A)$ is represented in unary, it follows that the number of sub-problems of the form $(r, succ^i(\gamma)) \models \psi$ to be considered is in $O(|A|^2)$.
- To decide if $(r, \gamma) \models \varphi$ holds, with φ of the form $K'\ \underline{\text{in}}\ \psi$, we need to decide whether $(r, [K' \to 0]\gamma) \models \psi$.
- To decide if $(r, \gamma) \models g$, we need to verify if the constraint g holds for the region γ. This can be done in constant time in the size of A because g only contains clocks in K, that is disjoint from the set of clocks C in the automaton A.

The depth of the recursive calls in this algorithm is at most $|\varphi|$, and each recursive call generates at most $O(|A|^2)$ subcalls. The computation of each recursive call takes time polynomial in $|\varphi|$, and thus constant in $|A|$. It follows that the time complexity of the aforementioned algorithm is in $O(|A|^{2|\varphi|})$ and, as φ is fixed, the program complexity is in P. □

Note that the aforementioned proofs of lower bounds for the specification complexity of model checking for L_ν and its fixed point free fragment are based upon the realization that the model checking problems of the form $nil \models \varphi$, where φ is a formula in any of the logics considered so far and nil is the (untimed) automaton with no transition, are just as hard as the model checking problems for arbitrary timed automata. We shall refer to this restricted model checking problem as "nil model checking". Indeed we have:

Lemma 13.
- *The nil model checking problem for L_ν, $L_{\mu,\nu}$ and $L^+_{\mu,\nu}$ is EXPTIME-complete.*
- *The nil model checking problem for L^-_ν is PSPACE-complete.*

Proof. That the nil model checking problem for $L^+_{\mu,\nu}$ (and, a fortiori, for L_ν and $L_{\mu,\nu}$) and L^-_ν are in EXPTIME and PSPACE, respectively, follows from Theorems 9 and 11. The proofs of the relevant lower bounds may be found in Appendix B. □

In light of the above results, the worst-case complexity of model checking for the real-time logics we have considered in this section 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 $L^-_{\forall S}$.

Remark 14. All the property languages we study in this paper, apart from $L_s + \langle a\rangle$, $L^-_s + \langle a\rangle$, $SBLL^-$ and $L^-_{\forall S}$, enjoy the following property:

> Given a timed automaton A and a formula φ, it is possible to build a formula $\varphi_{/A}$ s.t. $A \models \varphi$ iff $nil \models \varphi_{/A}$.

Thus the behaviour of a timed automaton A can be encoded, in some sense, in the formula $\varphi_{/A}$. More generally, these logics allow for *compositional model checking* [57,1,52,53]. Nevertheless this expressiveness result for these property languages cannot be used to deduce the computational hardness of the nil model checking problem from the complexity of the general model checking problem $A \models \varphi$, because the size of $\varphi_{/A}$ can be exponential in $|\varphi|$. For example, in the proof of the previous lemma, the PSPACE-hardness of the nil model checking problem for L^-_ν cannot be established by using the formula $\tilde{\Phi}_{/A_{(\varphi)}}$, where $A_{(\varphi)} \models \tilde{\Phi}$ is the L^-_ν model checking instance used in the proof of Theorem 11, because $|\tilde{\Phi}_{/A_{(\varphi)}}|$ is in $O(2^{|\tilde{\Phi}|})$.

Remark 15. Note that constructs of the form $K'\ \underline{\mathtt{in}}\ \varphi$, where K' is a set of formula clocks, were only employed in the lower bound proofs for the specification complexity of model checking of the logics considered in this section.

3.3. The property language L_s

The property language L_s (see Section 2.3) has been introduced in [57] 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 16. *The complexity of L_s model checking is PSPACE-complete. Moreover, the specification and program complexities of L_s model checking are also PSPACE-complete.*

Proof. PSPACE membership. A model checking problem of the form $A \models \varphi$, with $\varphi \in L_s$, can be reduced in polynomial time to a reachability problem for a parallel composition $(A|T_\varphi)$. See the more general Theorem 34 to follow.

PSPACE-hardness. We offer a linear time reduction from the reachability problem for timed automata, which is PSPACE-complete by Lemma 7, to L_s model checking.
Let A be a timed automaton and n one of its nodes. Build the automaton $\tilde{A}$ by relabelling all edges of A with a, and adding a new edge $\langle n, \mathtt{tt}, \mathtt{in\_n}, \emptyset, n\rangle$. Then n is reachable in A iff $\tilde{A} \not\models \mathtt{max}(X, [\mathtt{in\_n}]\mathtt{ff} \wedge [a]X \wedge \forall\!\!\forall X)$.

Specification complexity. It is possible to reduce, in polynomial time, any instance $\langle \mathscr{M}, w_0\rangle$ of the acceptance problem for linear bounded non-deterministic Turing machines to a model checking problem of the form $nil \models \Phi_{\mathscr{M},w_0}$ (with $\Phi_{\mathscr{M},w_0} \in L_s$) by means of the same kind of encoding used for L_ν (see Lemma 18 to follow).

Program complexity. It is PSPACE-complete because the formula expressing the reachability problem in the PSPACE-hardness argument above does not depend on the input automaton. $\square$

In light of the above result, 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. The source of the lower complexity is 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.

Theorem 17. *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.*

Proof. coNP membership. See the more general Theorem 35 to follow.

coNP-hardness. The *validity problem* for propositional formulae can be reduced, in polynomial time, to L_s^- model checking. Let ψ be a propositional formula over the set of atomic propositions $\{p_1, \ldots, p_n\}$. Consider the timed automaton $A_{(\neg\psi)}$ built from the negation of ψ as in Fig. 1. We claim that

$$\psi \text{ is valid iff } A_{(\neg\psi)} \models \forall\!\!\forall[a_1]\forall\!\!\forall[a_2]\cdots\forall\!\!\forall[a_n][sat]\mathtt{ff}.$$

To see that this does hold, consider an arbitrary path ρ in the TLTS $\mathscr{T}_{A_{(\neg\psi)}}$ of the form (2) on page 21. We argue that, if ψ is valid, its last configuration (r_n, u_n^ρ) satisfies the formula $[sat]\mathtt{ff}$. To this end, note that the configuration (r_n, u_n^ρ) is always reached at time n, and there are 2^n possible valuations u_n^ρ. They are of the form: $u_n^\rho(t) = n$ and $u_n^\rho(x_i) \in \{n - i, n\}$, for every $i \in \{1, \ldots, n\}$, depending on which clocks have been reset along the path. Since ψ is valid iff $\neg\psi$ is false for every interpretation of the p_i's, we deduce that ψ is valid iff $\widetilde{\neg\psi}$ (viz. the constraint on the values of the clocks x_i guarding the edge from node r_n to node end in $A_{(\neg\psi)}$) does not hold for any reachable (r_n, u_n^ρ). Thus $(r_n, u_n^\rho) \overset{sat}{\not\longrightarrow}$, which was to be shown.

In similar fashion, it is easy to argue that, if ψ is not valid, then there is a path in $\mathscr{T}_{A(\neg\psi)}$ of the form (2) whose last configuration (r_n, u_n^{ρ}) can perform a sat-labelled transition. Hence,

$$A_{(\neg\psi)} \not\models \forall\!\!\!\forall[a_1]\forall\!\!\!\forall[a_2]\cdots\forall\!\!\!\forall[a_n][sat]\mathtt{ff}$$

if ψ is not valid.

Specification complexity. Every instance of the validity problem for propositional formulae can be reduced, in polynomial time, to a model checking problem of the form $nil \models \Phi$ with $\Phi \in L_s^-$ (see Lemma 18 to follow).

Program complexity. This follows immediately from the fact that the program complexity for L_ν^- is in P (Theorem 11(3)). □

Lemma 18.
- *The nil model checking problem for L_s is PSPACE-complete.*
- *The nil model checking problem for L_s^- is coNP-complete.*

Proof. That the nil model checking problems for L_s and L_s^- are in PSPACE and coNP, respectively, follows from Theorems 16 and 17. The proofs of the relevant lower bounds may be found in Appendix B. □

3.4. Mapping the territory between L_s and L_ν

In Sections 3.2 and 3.3, we have shown, amongst other things, that the model checking problems for the property languages L_ν and L_s are EXPTIME-complete and PSPACE-complete, respectively. Since L_s is a restricted fragment of L_ν which has been proposed in [57] in order to improve the efficiency of model checking in practice, it is natural to wonder whether all the, seemingly ad hoc, syntactic restrictions imposed on L_s formulae are necessary in order to obtain a sublanguage of L_ν whose model checking problem is in PSPACE. To answer this natural question, we shall now consider several possible extensions of L_s. These extensions of L_s are obtained by adding one of the following constructs to that language:
- diamond modalities, viz. allowing for formulae like $\langle a\rangle\phi$,
- unrestricted disjunction, viz. allowing for formulae like $\psi \vee \phi$, and
- existential quantification over delay transitions, viz. allowing for formulae like $\exists\!\!\!\exists\phi$.

We consider these possible extensions in turn, and show that each of them leads to an EXPTIME-hard model checking problem. This implies that none of the restrictions imposed by the syntax for L_s can be dropped without increasing the structural complexity of the model checking problem.

3.4.1. Adding diamond modalities to L_s

Let $L_s + \langle a\rangle$ denote the property language obtained by adding the construct $\langle a\rangle\phi$ to the defining clauses for L_s. We shall now argue that:

Proposition 19. *The complexity of the model checking problem for $L_s + \langle a\rangle$ is EXPTIME-complete, and so are its program complexity and its specification complexity.*

Proof. We consider the combined, program, and specification complexities of model checking in turn.

Complexity of model checking. Since $L_s + \langle a\rangle$ is a sublanguage of L_ν, it is sufficient to argue that the complexity of the model checking problem for $L_s + \langle a\rangle$ is EXPTIME-hard. To this end, we offer a polynomial time reduction from the acceptance of an input w_0 by a LBATM $\mathscr{M}$ to a model checking problem $A_{\mathscr{M},w_0} \models \Phi'$ where $A_{\mathscr{M},w_0}$ is the timed automaton used in the proof of Lemma 7, and Φ' is an $L_s + \langle a\rangle$ formula used to encode the behaviour of $\mathscr{M}$ on input w_0. As in the proof of Theorem 9, we assume, without loss of generality, that we have a strict alternation of "or" and "and" states in $\mathscr{M}$; moreover we remind the reader that we assume that the initial and final states of $\mathscr{M}$ are "or" states.

The formula Φ (cf. Eq. (1) on page 19) used to handle the alternating behaviour of the LBATM $\mathscr{M}$ in the proof of Theorem 9 can be replaced by the following one, which contains neither existential quantifications over delay transitions nor general disjunction:

$$\Phi' = \forall\!\!\forall[s_0]\mathtt{max}\Big(X, [\mathtt{accept}]\mathtt{ff} \wedge \forall\!\!\forall\,[s]\ (z\ \underline{\mathtt{in}}\ \forall\!\!\forall(z = 1 \quad\Rightarrow\quad \langle s\rangle X))\Big).$$

Note that the existential quantification over delay transitions used in (1) (cf. page 19) may be replaced by the subformula

$$z\ \underline{\mathtt{in}}\ \forall\!\!\forall(z = 1 \quad\Rightarrow\quad \cdots)$$

because we know that exactly one time unit separates any two consecutive s-labelled transitions in the simulation of the workings of an LBATM on input w_0 by the timed automaton offered in the proof of Lemma 7. Again $\mathscr{M}$ accepts w_0 iff $A_{\mathscr{M},w_0} \not\models \Phi'$. This yields the EXPTIME-hardness of $L_s + \langle a\rangle$ model checking.

Program complexity. Since the formula Φ' we have constructed is fixed, and does not depend on the LBATM $\mathscr{M}$ and on w_0, the EXPTIME-hardness of the program complexity of the model checking problem also follows.

Specification complexity. See Appendix C. □

Remark 20. Unlike the general model checking problem, the nil model checking problem for $L_s + \langle a\rangle$ is still PSPACE-complete. This follows because the satisfaction of formulae involving the action modalities can be trivially checked for the nil process. Indeed, a model checking problem of the form $nil \models \phi$, with $\phi \in L_s + \langle a\rangle$, can be reduced in time linear in the size of ϕ to a model checking problem of the form $nil \models \tilde{\phi}$, where $\tilde{\phi}$ is the L_s-formula obtained from ϕ by replacing any subformula of the form $\langle a\rangle\varphi$ by $\mathtt{ff}$.

Remark 21. The extension of L_s with restricted diamond formulae of the form $\langle a\rangle\mathtt{tt}$ does not increase the complexity of the model checking problem, which remains PSPACE-complete as for L_s. Indeed, given a model checking problem $A \models \varphi$, where φ is a formula in L_s that can additionally use subformulae of the form $\langle a\rangle\mathtt{tt}$ ($a \in \mathsf{Act}$), it is possible to build in polynomial time a timed automaton $\tilde{A}$ and a formula $\tilde{\varphi} \in L_s$ such that $A \models \varphi$ iff $\tilde{A} \models \tilde{\varphi}$. The timed automaton $\tilde{A}$ is built by adding to A transitions of the form $(n, g_n, \mathtt{no\_a}, \emptyset, n)$ for every node n, where no_a is a fresh action symbol and

$$g_n = \neg\Bigg(\bigvee_{(n,g',a,r,n')\in A} g'\Bigg).$$

It is easy to see that $(n, u) \models \langle a \rangle \mathtt{tt}$ (in A) iff $(n, u) \models [\mathtt{no\_a}]\mathtt{ff}$ (in $\tilde{A}$), for every node n and clock valuation u. The formula $\tilde{\varphi}$ is built from φ by replacing every occurrence of $\langle a \rangle \mathtt{tt}$ ($a \in \mathsf{Act}$) with $[\mathtt{no\_a}]\mathtt{ff}$.

For the fragment of $L_s + \langle a \rangle$ without greatest fixed points, we have the following complexity results:

Proposition 22.
(1) *The model checking problem for $L_s^- + \langle a \rangle$ is PSPACE-complete.*
(2) *The specification complexity of $L_s^- + \langle a \rangle$ model checking is PSPACE-complete.*
(3) *The nil model checking problem for $L_s^- + \langle a \rangle$ is coNP-complete.*
(4) *The program complexity of $L_s^- + \langle a \rangle$ model checking is in P, if the constants in the input automata are represented in unary.*

Proof. We prove the four statements in turn.
(1) Since $L_s^- + \langle a \rangle$ is a sublanguage of L_ν^-, the model checking problem for $L_s^- + \langle a \rangle$ is in PSPACE by Theorem 11(1). To establish hardness for PSPACE, we argue that any instance of QBF can be reduced in polynomial time to a model checking problem for $L_s^- + \langle a \rangle$.
Let $\Phi = Q_1 p_1 \cdots Q_n p_n \cdot \varphi$ be an instance of the QBF problem. Consider the timed automaton $A_{(\varphi)}$ in Fig. 1. In the proof of Theorem 11(1), we argued that Φ is valid iff the timed automaton $A_{(\varphi)}$ satisfies the L_ν^- formula

$$\exists\!\!\exists(\langle a_1 \rangle \mathtt{tt} \wedge O_1(\exists\!\!\exists(\langle a_2 \rangle \mathtt{tt} \wedge O_2 \cdots \exists\!\!\exists(\langle a_n \rangle \mathtt{tt} \wedge O_n \langle sat \rangle \mathtt{tt})))),$$

where O_i is $\langle a_i \rangle$ (resp. $[a_i]$) if Q_i is $\exists$ (resp. $\forall$). This formula is not in $L_s^- + \langle a \rangle$ because it uses the existential delay operator $\exists\!\!\exists$. Note, however, that consecutive action transitions in $A_{(\varphi)}$ are separated by exactly one time unit. In lieu of the above formula, we can thus consider the following one, where t' is a fresh clock:

$$\tilde{\Phi} \overset{\text{def}}{=} t' \ \underline{\mathrm{in}} \ \forall\!\!\forall \Big(t' = 1 \ \Rightarrow \ O_1 \forall\!\!\forall(t' = 2 \ \Rightarrow \ \ldots O_{n-1} \forall\!\!\forall(t' = n \ \Rightarrow \ O_n \langle sat \rangle \mathtt{tt}))\Big),$$

where O_i is $\langle a_i \rangle$ (resp. $[a_i]$) if Q_i is $\exists$ (resp. $\forall$). We have that Φ is valid iff $A_{(\varphi)} \models \tilde{\Phi}$.
(2) See Appendix C for the proof of PSPACE-hardness of the specification complexity.
(3) This follows from Lemma 18 and Remark 20.
(4) This statement is an immediate corollary of Theorem 11 about L_ν^-. □

3.4.2. Adding disjunction to L_s

Let $L_s + \vee$ denote the property language obtained by adding the construct $\phi \vee \psi$ to the defining clauses for L_s. We shall now argue that:

Proposition 23. *The complexity of the (nil) model checking problem for $L_s + \vee$ is EXPTIME-complete, and so are its specification complexity and its program complexity.*

Proof. See Appendix C. □

For the fragment of $L_s + \vee$ without greatest fixed points, we have the following complexity results:

Proposition 24. *The model checking problem for $L_s^- + \vee$ is coNP-complete, and so are its specification complexity and the nil model checking problem. The program complexity is in P, if the constants in the input automata are represented in unary.*

Proof. The algorithm used in the proof of coNP membership for L_s^- model checking for concurrent programs (Theorem 35 to follow) can be easily adapted to $L_s^- + \vee$. Indeed such a non-deterministic polynomial algorithm can be used to decide if the negation of an L_s^- formula holds for a symbolic state $(\bar{r}, \gamma)$. To extend it to cover the whole of $L_s^- + \vee$, it is sufficient to add a clause for $(\bar{r}, \gamma) \models \psi_1 \wedge \psi_2$. In that case, the procedure simply checks whether $(\bar{r}, \gamma) \models \psi_1$ and $(\bar{r}, \gamma) \models \psi_2$ both hold. The resulting non-deterministic algorithm remains polynomial.

The coNP-hardness of (nil) model checking for $L_s^- + \vee$ is an immediate consequence of the coNP-hardness of nil model checking for L_s^- (Lemma 18). Since model checking for the nil process is coNP-complete, so is the specification complexity of model checking. The characterization of the program complexity comes from Theorem 11 about L_ν^-. $\square$

3.4.3. *Adding existential quantification over delay transitions to L_s*

Let $L_s + \exists$ denote the property language obtained by adding the construct $\exists\phi$ to the defining clauses for L_s. We shall now argue that:

Proposition 25. *The complexity of the (nil) model checking problem for $L_s + \exists$ is EXPTIME-complete, and so are its specification complexity and its program complexity.*

The proof is in Appendix C. For the fragment of $L_s + \exists$ without greatest fixed points, we have the following complexity results:

Proposition 26. *The model checking problem for $L_s^- + \exists$ is PSPACE-complete, and so are the nil model checking problem and the specification complexity. The program complexity is in P, if the constants in the input automata are represented in unary.*

Proof. The formula used in Appendix B to show that the nil-model checking problem for L_ν^- is PSPACE-hard belongs to $L_s^- + \exists$. The program complexity comes from Theorem 11 about L_ν^-. $\square$

Our main results mapping the territory between L_ν and L_s are summarized, for ease of reference, in Tables 4 and 5. It is interesting to note that adding $\langle a\rangle$ or $\exists$ increases the

Table 4
Complexity results for languages with fixed points

	L_s	$L_s + \langle a\rangle$	$L_s + \exists$	$L_s + \vee$	L_ν
MC, Spec. com., Prog. com.	PSPACE-C	EXPTIME-C	EXPTIME-C	EXPTIME-C	EXPTIME-C
	Theorem 16	Proposition 19	Proposition 25	Proposition 23	Theorem 9
nil − MC	PSPACE-C	PSPACE-C	EXPTIME-C	EXPTIME-C	EXPTIME-C
	Lemma 18	Remark 20	Proposition 25	Proposition 23	Lemma 13

Table 5
Complexity results for languages with fixed points

	L_s^-	$L_s^- + \langle a \rangle$	$L_s^- + \exists$	$L_s^- + \vee$	$L_\nu{}^-$
MC	coNP-C	PSPACE-C	PSPACE-C	coNP-C	PSPACE-C
Spec. compl.	coNP-C	PSPACE-C	PSPACE-C	coNP-C	PSPACE-C
Prog. compl.	P	P	P	P	P
	Theorem 17	Proposition 22	Proposition 26	Proposition 24	Theorem 11
nil – MC	coNP-C	coNP-C	PSPACE-C	coNP-C	PSPACE-C
	Lemma 18	Lemma 18	Proposition 26	Proposition 24	Lemma 13

complexity of model checking for L_s^-, whereas adding $\vee$ does not. Thus the addition of general disjunction to L_s only increases the complexity of model checking in the presence of fixed points.

3.5. The property languages $SBLL$ and $L_{\forall S}$

The model checking problems for property languages $SBLL$ and $L_{\forall S}$ have the same complexity, as stated in the following theorem, whose proof is similar to that of Theorem 16 and is therefore omitted:

Theorem 27. *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 $L_{\forall S}$ model checking are also PSPACE-complete.*

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

Theorem 28. *The model checking problem for $SBLL^-$ and $L_{\forall S}^-$ is PSPACE-complete. The specification complexity of model checking for $SBLL^-$ and $L_{\forall S}^-$ is coNP-complete. Finally, the program complexity of model checking for the property languages $SBLL^-$ and $L_{\forall S}^-$ is PSPACE-complete.*

Proof. PSPACE membership. This follows from the more general statement in Theorem 36.

PSPACE-hardness. The reachability problem can be reduced in linear time to a model checking problem for $SBLL^-$ (and then for $L_{\forall S}^-$). Given a timed automaton A, let r be the node we want to reach in A from the initial configuration. We build a timed automaton $\tilde{A}$ from A by replacing every transition label by τ, and by adding an action transition $r \xrightarrow{\mathtt{in\_r}} r$. The control node r is reachable iff $\tilde{A} \not\models \forall\!\!\!\forall[\mathtt{in\_r}]\mathtt{ff}$.

Specification complexity. Fix a timed automaton A. We can define a non-deterministic algorithm that, given a formula φ, decides in time polynomial in $|\varphi|$ whether $A \not\models \varphi$. Note that we can compute in constant time (w.r.t. to $|\varphi|$) the τ-derivatives and a-derivatives of any symbolic state (r, γ) of A, because this computation does not depend on the clocks in the formula φ. The algorithm uses the same ideas employed in the proof of Theorem 35 to follow. The hardness result is a consequence of Lemma 29 to follow, showing that the model checking for both $SBLL^-$ and $L^-_{\forall S}$ is coNP-hard if we fix the timed automaton to be the nil process.

Program complexity. As previously argued, checking whether a timed automaton satisfies the fixed formula $\forall\!\!\!\forall$`[in_r]`ff is PSPACE-hard. $\square$

The attentive reader might have noticed that the complexity of model checking for the fixed point-free versions of the languages $SBLL$ and $L_{\forall S}$ coincides with that for the full languages. This is because there is an implicit recursion (over τ and delay transitions) which is hidden in the semantics of the $SBLL^-$ operator $\forall\!\!\!\forall$. This recursion is sufficient to make the model checking problems for $SBLL^-$ and $L^-_{\forall S}$ PSPACE-hard, and is exploited in the aforementioned proof of the hardness result.

The nil model checking problems for $SBLL$ and $L_{\forall S}$ (resp. $SBLL^-$ and $L^-_{\forall S}$) are equivalent to the nil model checking problem for L_s (resp. L^-_s) because no τ-transition occurs in the nil process. For τ-free timed automata, the weak interpretation is equivalent to the strong interpretation. Moreover subformulae of the form $\langle a\rangle$tt can be replaced by ff because no action transition occurs in the nil process, and $\forall\!\!\!\forall_S$ is equivalent to $\forall\!\!\!\forall$. It thus follows from Lemma 18 that:

Lemma 29.
- *The nil model checking problem for $SBLL$ and $L_{\forall S}$ is PSPACE-complete.*
- *The nil model checking problem for $SBLL^-$ and $L^-_{\forall S}$ is coNP-complete.*

4. The complexity of model checking for concurrent timed programs

In practice real-time systems contain several components and are often modelled as *parallel compositions of timed automata*. We denote by *model checking for concurrent programs* the problem of deciding if a property holds for a concurrent timed system (viz. a parallel composition of timed automata). Since, already in the untimed setting, model checking suffers from the so-called state explosion problem in the presence of concurrency, a characterization of the complexity of the model checking problem for concurrent programs would yield a more realistic assessment of the hardness of the task of model checking real-time systems. The aim of this section is to present such an analysis.

Real-time concurrent programs. Let $A_1 \ldots, A_n$ be timed automata, with

$$A_i = \langle \mathsf{Act}, N_i, q_{i,0}, C_i, E_i\rangle.$$

(We assume that the sets of clocks $C_1, \ldots, C_n$ are pairwise disjoint.) We model a real-time concurrent program by means of the parallel composition of $A_1, \ldots, A_n$. Following [46], we use a parallel composition operator parameterized with a synchronization function. (Such an operator generalizes a large range of existing notions of parallel composition.) A *synchronization function* f is a partial function with signature $(\mathsf{Act} \cup \{\bullet\})^n \hookrightarrow \mathsf{Act}$, where $\bullet$ denotes a distinguished no-action symbol. In fact, f is an n-ary synchronization function with renaming. We denote by $(A_1|\cdots|A_n)_f$ the parallel composition of $A_1, \ldots, A_n$ with respect to the synchronization function f. A network configuration is a pair $(\bar{q}, v)$ where $\bar{q} = (q_1, \ldots, q_n)$ is a vector of nodes in $N_1 \times \cdots \times N_n$ and v is a valuation for $C = \bigcup_i C_i$, i.e., the set of clocks of the network. Given a network configuration $(\bar{q}, v)$, we write q_i for the ith node in the vector $\bar{q}$, and v_i for the restriction of v to the set of clocks C_i.

The semantics of $(A_1|\cdots|A_n)_f$ can be defined in terms of a timed labelled transition system, whose states are the configurations of the network and whose transitions are given by the two following rules:

- $(\bar{q}, v)\xrightarrow{b}(\bar{q'}, v')$ iff there is a $(a_1, \ldots, a_n) \in (\mathsf{Act} \cup \{\bullet\})^n$ such that:
  - for all $i \in \{1, \ldots, n\}$ with $a_i = \bullet$, we have that $q'_i = q_i$ and $v'_i = v_i$,
  - for all $i \in \{1, \ldots, n\}$ with $a_i \in \mathsf{Act}$, we have that $(q_i, v_i)\xrightarrow{a_i}(q'_i, v'_i)$, and
  - $f(a_1, \ldots, a_n) = b$;
- $(\bar{q}, v)\xrightarrow{\epsilon(d)}(\bar{q'}, v')$ iff $\bar{q'} = \bar{q}$ and $v' = v + d$.

Note that the second rule stipulates that all clocks increase synchronously.

When the synchronization function f is understood from the context, we simply write $\overline{A}$ in lieu of $(A_1|\cdots|A_n)_f$. The size of $\overline{A}$ is the sum of the sizes of the timed automata $A_1, \ldots, A_n$ and the size of the function f (viz. the number of synchronization vectors). Symbolically, $|\overline{A}| = |A_1| + \cdots + |A_n| + |f|$. Clearly $\overline{A}$ can be stored in space $\mathrm{O}(|\overline{A}|)$.

The possibility of forming parallel compositions of automata does not add expressive power to the model of timed automata. Indeed, from $(A_1|\cdots|A_n)_f$, it is possible to build a (product) timed automaton A which is strongly bisimilar (in the sense of, e.g., [54]) to $(A_1|\cdots|A_n)$. Such an automaton is $A = \langle \mathsf{Act}, N, \bar{q_0}, C, E\rangle$ with

- $N = N_1 \times \cdots \times N_n$,
- $\bar{q_0} = (q_{1,0}, \ldots, q_{n,0})$,
- $C = C_1 \cup \cdots \cup C_n$, and
- E contains an edge $\bar{q} \xrightarrow{g,a,r} \bar{q'}$ iff there are $a_1, \ldots, a_n \in \mathsf{Act} \cup \{\bullet\}$, guards $g_1, \ldots, g_n$ and reset sets $r_1, \ldots, r_n$ such that:
  - $f(a_1, \ldots, a_n) = a$, $g = \bigwedge_i g_i$ and $r = \bigcup_i r_i$, and
  - for every $i \in \{1, \ldots, n\}$,
    - $a_i = \bullet$, $g_i = \mathtt{tt}$, $r_i = \emptyset$ and $q'_i = q_i$, or
    - there is an edge $q_i \xrightarrow{g_i,a_i,r_i} q'_i$ in A_i.

It is well known (see, e.g., [35,39]) that the size of the automaton A so constructed is in $2^{\mathrm{O}(|\bar{A}|)}$. Moreover, as in the untimed case, it is not hard to prove that:

Lemma 30. *$\bar{A}$ and A satisfy exactly the same formulae in $L^{+}_{\mu,\nu}$.*

Therefore, to decide whether $(A_1|\cdots|A_n)_f \models \varphi$ holds for a formula $\varphi \in L^{+}_{\mu,\nu}$, it is sufficient to consider the region graph $(S, \longrightarrow)$ corresponding to φ and the product timed automaton A associated with the parallel composition $(A_1|\cdots|A_n)_f$. Using the analysis in Section 2.4, we have that:

$$S = N_1 \times \cdots \times N_n \times \mathscr{R}_M^{C^+},$$

where

- $M = \max(\mathtt{MCst}(A), \mathtt{MCst}(\varphi))$, and
- $C^+ = \bigcup_i C_i \cup K$.

(Recall that K denotes the set of formula clocks.) Moreover the size of the transition relation in the region graph, viz. $|\longrightarrow|$, is in

$$\mathrm{O}\Big((|f| \cdot \prod_i |E_i| + \prod_i |N_i|) \cdot |C^+|! \cdot M^{|C^+|}\Big).$$

Note, furthermore, that the size of the region graph associated with the product automaton A and the formula φ (viz. $|S| + |\rightarrow|$) is exponential in $(|\varphi| + |\bar{A}|)^2$.

For the sake of clarity, we remark that the regions used in algorithms for model checking networks of timed automata deal with the set of clocks $\bigcup_i C_i \cup K$. Moreover, we recall that the basic operations on regions mentioned in Section 2.4 can be performed in polynomial time.

We are now in a position to study the complexity of model checking for the property languages considered in the previous section over concurrent timed systems. We shall see that, in all cases, the concurrency feature does not increase the structural complexity of the model checking problems.

4.1. Reachability in concurrent timed systems

We begin by studying the complexity of the reachability problem in networks of timed automata. It is well known that reachability in communicating finite state machines is PSPACE-complete [49]. For parallel compositions of timed automata, we obtain the following result.

Theorem 31. *The node reachability problem over concurrent timed systems is PSPACE-complete.*

Proof. PSPACE membership. Let $\overline{A} = (A_1 | \cdots | A_n)_f$ be a parallel composition of timed automata. Let r be a node of A_i ($i \in \{1, \ldots, n\}$). Deciding whether r is reachable from the initial state $(\bar{q_0}, \gamma_0)$ of the region graph associated to the product automaton A determined by $\bar{A}$ (which is equivalent to the reachability of r in the timed labelled transition system associated with $\bar{A}$) can be done by using the following non-deterministic algorithm which builds, step by step, a path in the region graph leading to a configuration with r as one of its control nodes, if one such configuration and path exist:

> If the current state $(\bar{q}, \gamma)$ contains r then the answer is "yes", otherwise the algorithm guesses $(\bar{q'}, \gamma')$, that is the next configuration on a path leading to r, where $\bar{q'} \in N_1 \times \cdots \times N_n$ and γ is a region over $C_1 \cup \cdots \cup C_n$, and it verifies (in polynomial time) that $(\bar{q}, \gamma) \xrightarrow{succ^*} (\bar{q'}, \gamma')$ or $(\bar{q}, \gamma) \xrightarrow{a} (\bar{q'}, \gamma')$, for some action a.

At any step we only need to store at most two configurations. By Savitch's theorem [66], the use of non-determinism in the above algorithm is inessential.

PSPACE-hardness. The reachability problem in a (single) timed automaton is PSPACE-hard (see Lemma 7). □

4.2. Model checking for sublanguages of $L^+_{\mu,\nu}$ for concurrent timed systems

The (program) complexity of model checking for the (alternation-free) μ-calculus for untimed *concurrent* programs is EXPTIME-complete [51]. For parallel compositions of timed automata and fragments of $L^+_{\mu,\nu}$, we obtain the following results.

Theorem 32. *The model checking problem for $L^+_{\mu,\nu}$, $L_{\mu,\nu}$ and L_ν over concurrent timed systems is EXPTIME-complete.*

Proof. EXPTIME membership. We use an approach similar to the one used in [51] to prove the EXPTIME membership of the (untimed) μ-calculus model checking problem for concurrent programs. We have previously seen in Lemma 30 that, for every formula $\varphi \in L^{+}_{\mu,\nu}$,

$$(A_1|\cdots|A_n)_f \models \varphi \text{ iff } A \models \varphi,$$

where A is the timed automaton corresponding to the product of $(A_1|\cdots|A_n)_f$ with respect to the specified synchronization function. Moreover we know that $A \models \varphi$ iff $\tilde{A} \models \varphi$ where $\tilde{A}$ is an untimed automaton (the *region graph*) associated with A and φ whose size is exponential in $(|\varphi| + \sum_i |A_i| + |f|)^2$, and over which φ is interpreted as an untimed formula. If we modify slightly $\tilde{A}$ by adding the transitive closure of the transition relation $\xrightarrow{succ}$, the size of the resulting automaton is still exponential in $(|\varphi| + \sum_i |A_i| + |f|)^2$, and $\exists\!\!\exists$ and $\forall\!\!\forall$ become "one step" modalities. Then φ may be viewed as an untimed μ-calculus formula for which time complexity of model checking is in $O((|\varphi| \cdot |\tilde{A}|)^{alt+1})$ where alt is the alternation depth of φ [38]. Clearly we have $alt < |\varphi|$, and then we obtain an algorithm which is exponential in $|\varphi| \cdot (|\varphi| + \sum_i |A_i|)^2$. This gives the EXPTIME membership for $L^{+}_{\mu,\nu}$, $L_{\mu,\nu}$ and L_ν.

EXPTIME-hardness. The model checking problem for L_ν is EXPTIME-hard for one timed automaton (Theorem 9). □

Theorem 33. *The model checking problem for L^{-}_{ν} over concurrent timed systems is PSPACE-complete.*

Proof. PSPACE membership. A non-deterministic model checking algorithm in PSPACE can be easily defined by considering the parts of the region graph associated to $(A_1|\cdots|A_n)_f \models \varphi$ only when they are required. To this end, simply adapt the algorithm given in the proof of Theorem 11(3). Again, by Savitch's theorem [66], the use of non-determinism in the above algorithm is inessential.

PSPACE-hardness. Immediate by Theorem 11. □

Theorem 34. *The model checking problem for L_s, $SBLL$ and $L_{\forall S}$ over concurrent timed systems is PSPACE-complete.*

Proof. PSPACE membership. For any of these property languages, an instance of the model checking problem $(A_1|\cdots|A_n)_f \models \varphi$ can be reduced (in polynomial time) to a reachability problem [2,1] in $((A_1|\cdots|A_n)_f|T_\varphi)$ where T_φ is a timed automaton s.t. $|T_\varphi| \in O(|\varphi|)$. Hence Theorem 31 gives the PSPACE upper bound.

PSPACE-hardness. This is a consequence of Theorem 16. □

Theorem 35. *The model checking problem for L^{-}_{s} over concurrent timed systems is coNP-complete.*

Proof. coNP membership. Let $\bar{A} = (A_1|\cdots|A_n)_f$ be a concurrent timed system and ψ be an L^{-}_{s} formula. We can define a non-deterministic and "on the fly" polynomial algorithm over the region graph associated with A (viz. the product automaton induced by $\bar{A}$) to decide if the negation of ψ (containing only existential modalities) holds for $(\bar{n_0}, \gamma_0)$, the initial symbolic configuration of A. We give the main cases:

- CASE: $(\bar{r}, \gamma) \models \varphi_1 \vee \varphi_2$. The procedure first guesses non-deterministically the φ_i which has to hold for $(\bar{r}, \gamma)$, and then proceeds by checking whether $(\bar{r}, \gamma) \models \varphi_i$.
- CASE: $(\bar{r}, \gamma) \models g \wedge \varphi$. The procedure first verifies that g holds for region γ, and then proceeds by checking whether $(\bar{r}, \gamma) \models \varphi$.
- CASE: $(\bar{r}, \gamma) \models \exists\varphi$. First the procedure guesses a region γ' such that γ' is reachable from γ with a sequence of delay transitions. Then it verifies, in polynomial time, that $\gamma \xrightarrow{succ}{}^{*} \gamma'$ holds, and finally it checks whether $(\bar{r}, \gamma') \models \varphi$.
- CASE: $(\bar{r}, \gamma) \models \langle a\rangle\varphi$. The procedure guesses a configuration $(\bar{r'}, \gamma')$ such that $(\bar{r}, \gamma) \xrightarrow{a} (\bar{r'}, \gamma')$ holds (this can be verified in polynomial time), and then proceeds by checking whether $(\bar{r'}, \gamma') \models \varphi$.
- CASE: $(\bar{r}, \gamma) \models K' \; \underline{\text{in}} \; \varphi$. We simply proceed by checking whether it holds that $(\bar{r}, [K' \rightarrow 0]\gamma) \models \varphi$.

Note that the absence of fixed points entails that deciding whether $\bar{A}$ satisfies an L_s^- formula needs to consider only executions with at most $|\varphi|$ action transitions. For a formula φ, the algorithm sketched above takes at most $|\varphi|$ steps, and each step has a complexity polynomial in $(|\varphi| + |A|)$.

coNP-hardness. This is an immediate consequence of Theorem 17. □

The following result is a straightforward corollary of the PSPACE membership of reachability in networks of timed automata (Theorem 31) and of the PSPACE-hardness of the model checking problem for $SBLL^-$ and $L_{\forall S}^-$ (Theorem 28).

Theorem 36. *The model checking problem for $SBLL^-$ and $L_{\forall S}^-$ over concurrent timed systems is PSPACE-complete.*

These results show that, unlike in the untimed setting [51], the concurrency feature does not increase the structural complexity of model checking for timed systems. A natural question to ask is whether the same holds true for the specification and program complexities.

We have the following results:

- The specification complexity does not change because, since the program being considered is fixed, its structure does not matter (from a parallel composition of timed automata $\bar{A}$, it is possible to build the corresponding product timed automaton A, in time constant in the size of the specification).
- The program complexity for model checking with respect to networks of timed automata cannot change for $L_{\mu,\nu}^+$, $L_{\mu,\nu}$, L_ν, L_s, $SBLL$, $SBLL^-$, $L_{\forall S}$ and $L_{\forall S}^-$ since the program complexity for these property languages is already as hard for one timed automaton as the general model checking problem for concurrent timed systems.
  The program complexity for model checking L_ν^- over concurrent timed systems is NP-hard and coNP-hard (and belongs to PSPACE) since this is already the case for the untimed Hennessy–Milner logic: it is not hard to see that the satisfiability (resp. validity) problem for boolean expressions can be encoded as a model checking problem for the formula $\langle a\rangle[b]\, \mathtt{ff}$ (resp. $[a]\,[b]\, \mathtt{ff}$) over a parallel composition of automata.
  The program complexity for model checking L_s^- over concurrent timed systems is coNP-complete. Indeed coNP-hardness comes from the above remark, and coNP-membership can be shown by using similar arguments to those used in the proof of

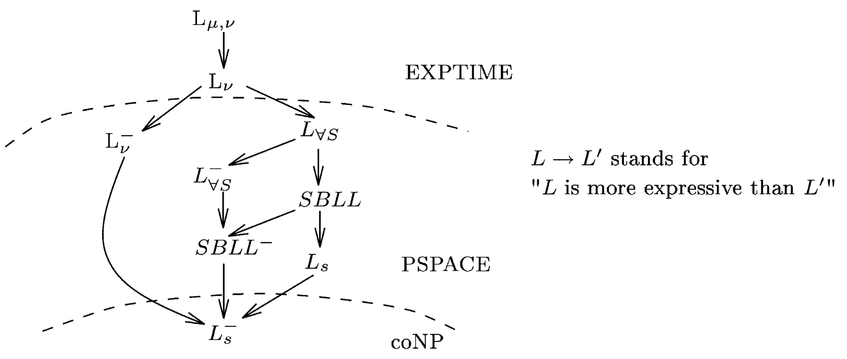

Fig. 2 Expressiveness vs complexity of model checking.

Theorem 11(3): if the formula Φ does not hold for a system S, then there exists a short path ρ (the number of action transitions is at most $|\Phi|$) in S satisfying $\neg\Phi$ and verifying $\rho \models \neg\Phi$ can be done in polynomial time whenever ρ is given.

5. Conclusion

The relationships between the relative expressive power of the property languages that we have considered in this paper, and the complexity of their model checking problems, are summarized in Fig. 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 an equivalent model checking problem $\tilde{A} \models \tilde{\varphi}$ with $\tilde{\varphi} \in L$.)

In the discussion to follow, we shall use the abbreviations $\mathscr{A}_{+t} \overset{?}{\models} \Psi_{+t}$, $\mathscr{A} \overset{?}{\models} \Psi_{+t}$ and $\mathscr{A}_{+t} \overset{?}{\models} \Psi$ 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.

Note that, for each of the specification languages that we have studied, the proof of $\mathscr{C}$-hardness of the corresponding model checking problem uses formulae without clocks (cf. the proofs of the hardness results in Theorems 9–28). This implies that the problems $\mathscr{A}_{+t} \overset{?}{\models} \Psi$ and $\mathscr{A}_{+t} \overset{?}{\models} \Psi_{+t}$ have the same complexity. The results on the complexity of model checking problems of the form $nil \models \varphi$ show that the problems $\mathscr{A} \overset{?}{\models} \Psi_{+t}$ and $\mathscr{A}_{+t} \overset{?}{\models} \Psi_{+t}$ also have the same complexity, with the exception of $L_s + \langle a \rangle$ and $L_s^{-} + \langle a \rangle$. Therefore the complexity of model checking does not depend on whether time is added to the model, to the specification or to both. Time seems to be a highly expressive feature in itself, as witnessed, e.g., by the encodings offered in Appendix B.

Our results on the structural complexity of model checking for real-time concurrent programs match those for untimed concurrent programs offered in, e.g., [51]. However, we think that these kind of results must be taken with a grain of salt, and cannot be used by themselves to claim that model checking real-time programs is just as efficient (or inefficient) as model checking untimed ones in practice. First of all, the worst case time complexity of model checking algorithms for an untimed specification logic like the μ-calculus interpreted w.r.t. concurrent programs is in $O(2^n)$, where n is the size of the input. On the other hand, the theoretical algorithms we have offered in this paper for model checking the logic $L^{+}_{\mu,\nu}$ w.r.t. networks of timed automata run in time $O(2^{n^2})$. In practice, there is a big difference between the two bounds. Secondly, and more importantly, the complexity of model checking real-time programs is due to at least two concomitant factors. Indeed, apart from the well-known state explosion problem brought about by the concurrency feature, one has to deal with the complexity deriving from the use of timing information (e.g., in the form of clocks and comparisons amongst clocks). The research community on verification technology has developed data structures (e.g., Reduced Ordered Binary Decision Diagrams [21]) and a variety of techniques (see, e.g., [13,20,26]) to alleviate the explosion in the number of control states deriving from cooperative concurrency. Similarly, efficient data structures have been developed to handle timing information (e.g., DBMs [33]). However, the development of efficient data structures and techniques to handle at the same time both timing information *and* the state explosion problem has so far proved elusive. We expect, however, exciting developments on this line of research—see, e.g., [16].

Acknowledgements

Carsten Weise suggested the title of this paper, and gave useful suggestions on a draft of its extended abstract [4]. Special thanks to Anders Ravn for his insightful comments and thorough proofreading. Many thanks to Philippe Schnoebelen for discussions about complexity and model checking. The four anonymous referees gave us many insightful comments on, and spotted a few inaccuracies in, the original version of this work. The responsibility for any remaining infelicity rests with the authors.

Appendix

A. The property languages *SBLL* and $L_{\forall\!\!\forall S}$

Let $ALW_\tau(\psi)$ be the formula $\mathtt{max}(X, \psi \wedge [\tau]X)$ (where X is a new identifier). Intuitively $ALW_\tau(\psi)$ holds for an extended state (n, v, u) of a timed automaton A iff ψ holds for any (n', v', u) whenever (n', v') is reachable from (n, v) in A by using only τ-transitions.

Let $ALW_{\tau,\delta}(\psi)$ be the formula $\mathtt{max}(X, \psi \wedge [\tau]X \wedge \forall\!\!\forall X)$ (where X is a new identifier). Intuitively, $ALW_{\tau,\delta}(\psi)$ holds for an extended state (n, v, u) of A iff ψ holds for any $(n', v', u + d)$ whenever (n', v') is reachable from (n, v) in A by using only τ-transitions and delay transitions of total duration d.

The rules in Table 6 give a way of building the equivalent L_ν formula $\overline{\varphi}$ from an $SBLL$ formula φ (for example with a top–down algorithm), and we have:

Table 6
Building $\overline{\varphi}$ from an $SBLL$ formula φ

$\overline{\mathtt{ff}}$	$\overset{\text{def}}{=} \mathtt{ff}$	$\overline{\psi_1 \wedge \psi_2}$	$\overset{\text{def}}{=} ALW_\tau(\overline{\psi_1} \wedge \overline{\psi_2})$
$\overline{x \ \underline{\mathtt{in}}\ \psi}$	$\overset{\text{def}}{=} x \ \underline{\mathtt{in}}\ ALW_\tau(\overline{\psi})$	$\overline{g \vee \psi}$	$\overset{\text{def}}{=} g \vee ALW_\tau(\overline{\psi})$
$\overline{[a]\psi}$	$\overset{\text{def}}{=} ALW_\tau([a]\overline{\psi})$	$\overline{\langle a\rangle \mathtt{tt}}$	$\overset{\text{def}}{=} ALW_\tau(\langle a\rangle \mathtt{tt})$
$\overline{\forall\!\!\forall\psi}$	$\overset{\text{def}}{=} ALW_{\tau,\delta}(\overline{\psi})$	$\overline{\mathtt{max}(Y, \psi)}$	$\overset{\text{def}}{=} ALW_\tau(\mathtt{max}(Y, \overline{\psi})$

Lemma A.1. *Let A be a timed automaton. Suppose that φ is an $SBLL$ formula, and let $\overline{\varphi}$ be the formula defined as above. Then we have that $A \models \varphi$ iff $A \models \overline{\varphi}$.*

The size of $\overline{\varphi}$ is linear in the size of φ. Note that it would be possible to translate every $SBLL$ formula into an extension of the property language L_s with formulae of the form $\langle a \rangle \mathtt{tt}$.

As for $SBLL$, it is possible to translate any $L_{\forall S}$ formula φ into a L_ν formula $\overline{\varphi}$ such that for any timed automaton A, we have $A \models \varphi \;\Leftrightarrow\; A \models \overline{\varphi}$. The only difference with $SBLL$ concerns the $\forall\!\!\forall_S$ operator:

$$\overline{\forall\!\!\forall_S \varphi} \stackrel{\text{def}}{=} \mathtt{max}\Bigl(X, \overline{\varphi} \wedge [\tau]X \wedge \Bigl(\bigvee_{a \in S} \langle a \rangle \mathtt{tt} \vee \forall\!\!\forall X\Bigr)\Bigr).$$

B. Complexity of *nil* model checking

This appendix is devoted to the proofs of the lower bounds for the complexity of the *nil* model checking problems for the logics studied in Section 3.2 of the main body of the paper.

Proof (of Lemma 13).

- **EXPTIME-hardness of *nil*-MC for $L^+_{\mu,\nu}$, $L_{\mu,\nu}$ and L_ν.** We show how we can encode the behaviour of an LBATM $\mathscr{M}$ over an input w_0 by means of an L_ν formula. The encoding we present uses formula clocks to represent the configurations of $\mathscr{M}$. Clock constraints and the $\underline{\mathtt{in}}$ operator are employed to simulate the transitions of $\mathscr{M}$ on input w_0. Again we assume that we have strict alternations of "and" and "or" states and that $\Sigma = \{a, b\}$. Our aim is to construct an L_ν formula Φ from the description of $\mathscr{M}$ and the input w_0 such that $\mathscr{M}$ accepts w_0 iff $nil \not\models \Phi$.
  In the formula Φ, we use $3n + k + 1$ clocks where $n = |w_0|$ and k is the number of states in $\mathscr{M}$. Furthermore, we assume that the states are numbered $q_0, \ldots, q_{k-1}$ with $q_F = q_{k-1}$. The set of clocks is

  $$\{x_i, y_i, c_i \mid i = 1, \ldots, n\} \cup \{z_j \mid j = 0, \ldots, k-1\} \cup \{t\}.$$

  The clocks x_i and y_i encode the contents of the ith tape cell thus: the ith tape cell contains the symbol a (resp. b) if $x_i = y_i$ (resp. $x_i < y_i$). The clocks c_i encode the position of the tape head: the head is at position i iff $c_i < 1$. The clocks z_j encode the current control state (when a transition has to be performed) as follows: $\mathscr{M}$ is in state q_j iff $z_j < 1$. The clock t is used to ensure time elapsing, and, indirectly, that after each transition there are exactly one clock c_i and one clock z_j whose values are strictly smaller than 1.
  Writing the initial configuration on the tape consists in waiting for some positive delay greater than 1, then resetting each $x_i \in r_{w_0}$, with

  $$r_{w_0} = \{x_i \mid w_0(i) = b\},$$

  (we write w_0 on the tape) and waiting for some positive delay, and thereafter resetting z_0 ($\mathscr{M}$ starts computing from its initial state q_0) and c_1 (the tape head is reading the first tape cell). We have:

  $$\Phi \stackrel{\text{def}}{=} \forall\!\!\forall\Bigl(t > 1 \Rightarrow \Bigl(r_{w_0} \cup \{t\} \;\underline{\mathtt{in}}\; \forall\!\!\forall(t > 0 \Rightarrow (\{z_0, c_1\} \;\underline{\mathtt{in}}\; \mathtt{max}(X, \Psi)))\Bigr)\Bigr). \qquad \text{(B.1)}$$

The formula $\mathtt{max}(X, \Psi)$ has to express that from an "or" state q, whatever is the chosen enabled transition in $\mathscr{M}$ which leads from q to an "and" state q', it is then possible to find a successor configuration that is not accepting (that is, which has $z_{k-1} \geq 1$) and so on. For every transition $\theta = (q_j, \alpha, \alpha', \delta, q_{j'})$ of $\mathscr{M}$ with q_j an *or*-state, let φ^i_θ be the following formula which simulates the transition θ when the tape head is reading cell i:

$$\varphi^i_\theta \overset{\text{def}}{=} (z_j < 1 \wedge c_i < 1 \wedge g^i_\alpha) \Rightarrow$$
$$r^i_{\alpha'} \cup \{t\} \ \underline{\mathtt{in}}\ \forall\!\!\!\forall\Big(t > 1 \Rightarrow (\{z_{j'}, c_{i'}\} \ \underline{\mathtt{in}} \bigvee_{\theta' \in T(q_{j'})} \psi^{i'}_{\theta'})\Big), \tag{B.2}$$

where $T(q_{j'})$ denotes the collection of transitions whose source is state $q_{j'}$. The guard g^i_α is $x_i = y_i$ (resp. $x_i < y_i$) if α is a (resp. b). The reset set $r^i_{\alpha'}$ is $\{x_i, y_i\}$ (resp. $\{x_i\}$) if α' is a (resp. b). Finally $i' = i + 1$ if $\delta = R$ and $i < n$, and $i' = i - 1$ if $\delta = L$ and $i > 1$. The formula φ^i_θ can be read as follows: "if the current state is q_j, the tape head is at i and the ith symbol of the tape is α, then after the transition θ, there exists a transition θ' starting from the new configuration (control state $q_{j'}$, symbol α' at i and i' as new position for the head) such that $\psi^{i'}_{\theta'}$ holds". For every *and*-state q_j, the formula ψ^i_θ states that transition $\theta = (q_j, \alpha, \alpha', \delta, q_{j'})$ leads from a configuration whose state is q_j and with tape head at i to a configuration in X (that is a non-accepting *or*-configuration) thus:

$$\psi^i_\theta \overset{\text{def}}{=} g^i_\alpha \wedge \Big(r^i_{\alpha'} \cup \{t\} \ \underline{\mathtt{in}}\ \forall\!\!\!\forall\big(t > 1 \Rightarrow \{z_{j'}, c_{i'}\} \ \underline{\mathtt{in}}\ X\big)\Big). \tag{B.3}$$

Note that $z_{j'}$ and $c_{i'}$ are reset after the delay making $t > 1$ in order to ensure that their value will be 0 (and thus less than 1) for the next step of $\mathscr{M}$. Finally the formula Ψ in (B.1) is the following one:

$$\Psi \overset{\text{def}}{=} (z_{k-1} \geqslant 1) \wedge \bigwedge_{i=1}^{n} \bigwedge_{\theta \in T_{\text{or}}} \varphi^i_\theta, \tag{B.4}$$

where T_{or} is the set of transitions starting from an *or*-state. The LBATM $\mathscr{M}$ accepts w_0 iff $nil \not\models \Phi$. The reduction we have just presented is polynomial, and gives the EXPTIME-hardness of the nil model checking problem for L_ν (and, *a fortiori*, for $L^+_{\mu,\nu}$ and $L_{\mu,\nu}$).

- **PSPACE-hardness of *nil*-MC for L^-_ν.** Let $\Phi = Q_1 p_1 \cdots Q_n p_n \cdot \varphi$ be an instance of the QBF problem. The following formula $\bar{\Phi}$ encodes it as a model checking problem for nil:

  $\{t, x_1, \ldots, x_n\} \ \underline{\mathtt{in}}\ \exists\!\!\!\exists \ldots$

$$\exists\!\!\!\exists\Bigg(t = i \wedge \left| \begin{array}{l} \exists\!\!\!\exists\,\big(t < i + 1 \wedge \\ \forall\!\!\!\forall\big(t < i + 1 \Rightarrow \end{array} \right| \ x_i \ \underline{\mathtt{in}}\ \exists\!\!\!\exists(t = i + 1 \wedge \ldots \exists\!\!\!\exists(t = n + 1 \wedge \bar{\varphi}))\big)\Bigg)$$

  The first (respectively, second) line corresponds to the case $Q_i = \exists$ (resp. $Q_i = \forall$). Moreover,

$$\bar{\varphi} \overset{\text{def}}{=} \varphi[p_i \leftarrow x_i = n + 1 - i,\ \bar{p}_i \leftarrow x_i < n + 1 - i]^n_{i=1}.$$

Note that each clock x_i is reset to 0 at a time in $[i; i+1[$. The interpretation of the above construction is as follows: if x_i is reset at i, then p_i is $\mathtt{tt}$, otherwise p_i is $\mathtt{ff}$. We have that $nil \models \bar{\Phi}$ iff Φ is valid. □

Proof (of Lemma 18).

- **PSPACE-hardness of** nil**-MC for** L_s**.** We show how we can encode the behaviour of a linear bounded non-deterministic Turing machine $\mathscr{M}$ over an input w_0 of length n as an L_s formula Φ which holds for nil iff w_0 is not accepted by $\mathscr{M}$. To this end, we use the same idea[4] underlying the proof of EXPTIME-hardness of nil-MC for L_ν (see the proof of Lemma 13). Consider the formula:

$$\Phi \stackrel{\text{def}}{=} \forall\!\!\forall\Big(t > 1 \Rightarrow \Big(r_{w_0} \cup \{t\} \;\underline{\mathtt{in}}\; \forall\!\!\forall(t > 0 \Rightarrow (\{z_0, c_1\} \;\underline{\mathtt{in}}\; \mathtt{max}(X, \Psi)))\Big)\Big).$$

We remark that the formula Ψ used for proving Lemma 13 encodes an alternating behavior of $\mathscr{M}$ and does not belong to L_s.
Here $\mathscr{M}$ is not an alternating machine and the formula $\mathtt{max}(X, \Psi)$ has just to express that whatever the chosen transition in $\mathscr{M}$, it is not possible to reach an accepting configuration (that is, $z_{k-1} \geq 1$ always holds). For every transition $\theta = (q_j, \alpha, \alpha', \delta, q_{j'})$ of $\mathscr{M}$, let φ^i_θ be the following formula which simulates the transition θ when the tape head is reading cell i:

$$\varphi^i_\theta \stackrel{\text{def}}{=} (z_j < 1 \wedge c_i < 1 \wedge g^i_\alpha) \;\Rightarrow\; r^i_{\alpha'} \cup \{t\} \;\underline{\mathtt{in}}\; \forall\!\!\forall\Big(t > 1 \;\Rightarrow\; (\{z_{j'}, c_{i'}\} \;\underline{\mathtt{in}}\; X)\Big),$$

where g^i_α is $x_i = y_i$ (resp. $x_i < y_i$) if α is a (resp. b). The reset set $r^i_{\alpha'}$ is $\{x_i, y_i\}$ (resp. $\{x_i\}$) if α' is a (resp. b). Finally $i' = i+1$ if $\delta = R$ and $i < n$, and $i' = i-1$ if $\delta = L$ and $i > 1$. The formula Ψ is the following one:

$$\Psi \stackrel{\text{def}}{=} (z_{k-1} \geqslant 1) \wedge \bigwedge_{\theta, i} \varphi^i_\theta.$$

The formula Φ belongs to L_s and its size is in $\mathrm{O}(|\mathscr{M}| \cdot |w_0|)$. The construction is done in polynomial time and we have that $nil \models \Phi$ iff q_{k-1} is not reachable in $\mathscr{M}$ on input w_0.
- **coNP-hardness of** nil**-MC for** L^-_s**.** In the proof of the second statement of Lemma 13, we encoded an instance of the problem QBF as an L^-_ν formula which has to hold for the nil process. We can transform this formula into an L^-_s formula encoding a validity problem. Note that a validity problem for φ over the atomic propositions $\{p_1, \ldots, p_n\}$ can be seen as the following QBF problem: $\forall p_1.\forall p_2 \ldots \forall p_n.\varphi$. We can reuse the formula Φ used for proving Lemma 13 with small changes in order to obtain an L^-_s formula. To this end, we replace the subformulae $\exists\!\!\exists(t = i \wedge \ldots)$ by $\forall\!\!\forall(t = i \;\Rightarrow\; \ldots)$. With this modification, we obtain $\Phi = \{t, x_1, \ldots, x_n\} \;\underline{\mathtt{in}}\; \Psi$ with:

$$\Psi = \begin{pmatrix} \forall\!\!\forall \cdots \;\forall\!\!\forall\Big(t = i \;\Rightarrow\; \forall\!\!\forall\big(t < i+1 \;\Rightarrow\; x_i \;\underline{\mathtt{in}}\; \forall\!\!\forall(t = i+1 \;\Rightarrow\; \cdots \\ \cdots \quad \forall\!\!\forall(t = n+1 \;\Rightarrow\; \bar{\varphi}))\big)\Big) \end{pmatrix},$$

where $\bar{\varphi} = \varphi[p_i \leftarrow x_i = n+1-i, \bar{p}_i \leftarrow x_i < n+1-i]^n_{i=1}$. □

[4] The same encoding is used, but, since $\mathscr{M}$ is not an alternating machine, every control state is an "or" state.

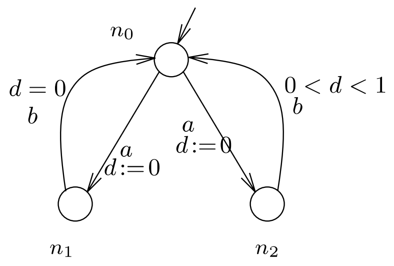

Fig. 3. The timed automaton $\mathscr{A}$.

C. The extensions of L_s

This appendix is devoted to proofs that were omitted from Section 3.4 in the main body of the paper.

Proof (of Proposition 19). We show the EXPTIME-hardness of specification complexity for $L_s + \langle a \rangle$ model checking. We reduce the acceptance problem of an input word w_0 by a LBATM $\mathscr{M}$ to a model checking instance $A \models \varphi$ where A is a timed automaton which does not depend on $\mathscr{M}$ or w_0, and $\varphi \in L_s + \langle a \rangle$.

We assume, without loss of generality, that for each control state q, there are two transitions ($\theta^q_{a,1}$ and $\theta^q_{a,2}$) of the form $(q, a, \ldots)$ and two transitions ($\theta^q_{b,1}$ and $\theta^q_{b,2}$) of the form $(q, b, \ldots)$. Therefore for any configuration of $\mathscr{M}$, there are exactly *two* enabled transitions.

We reuse the encoding done to prove EXPTIME-hardness of nil-model checking for L_ν (Lemma 13, Appendix B): we use $2n$ clocks ($|w_0| = n$) x_i and y_i ($1 \leqslant i \leqslant n$) to represent the tape content, n clocks c_i ($1 \leqslant i \leqslant n$) to store the current position of the tape head, k clocks z_j (k is the number of control states of $\mathscr{M}$). Moreover we use two additional clocks t and t'. Let $\mathscr{A}$ be the timed automaton described in Fig. 3. From the node n_0, there are two a-transitions: the first one has to be followed by a b-transition with a zero delay while the second one requires a delay in $]0;1[$ before performing the b-transition. We use this fact to encode the existential choice. $\mathscr{M}$ accepts w_0 iff $\mathscr{A} \not\models \Phi$ with

$$\Phi \stackrel{\text{def}}{=} \forall\!\!\!\forall\Big(t > 1 \;\Rightarrow\; \Big(r_{w_0} \cup \{t\}\ \underline{\mathtt{in}}\ \forall\!\!\!\forall(t > 0 \;\Rightarrow\; \{z_0, c_1\}\ \underline{\mathtt{in}}\ \mathtt{max}(X, \Psi))\Big)\Big),$$

$$\Psi \stackrel{\text{def}}{=} (z_{k-1} \geqslant 1) \wedge \bigwedge_{i=1}^{n} \bigwedge_{\theta \in T_{\text{or}}} \varphi^i_\theta.$$

Let θ be a transition $(q_j, \alpha, \alpha', \delta, q_{j'})$. We have

$$\begin{aligned}\varphi^i_\theta \stackrel{\text{def}}{=} {} & (z_j < 1) \wedge (c_i < 1) \wedge g^i_\alpha \;\Rightarrow\; \Bigg[(r^i_{\alpha'} \cup \{t\})\ \underline{\mathtt{in}}\ \forall\!\!\!\forall\Bigg(t > 1 \;\Rightarrow\; \{t'\}\ \underline{\mathtt{in}} \\ & \langle a \rangle \forall\!\!\!\forall[b]\Bigg[t' = 0 \;\Rightarrow\; \{z_{j'}, c_{i'}\}\ \underline{\mathtt{in}}\ \Bigg((x_{i'} = y_{i'} \;\Rightarrow\; \psi^{i'}_{\theta^{q_{j'}}_{a,1}}) \wedge (x_{i'} < y_{i'} \;\Rightarrow\; \psi^{i'}_{\theta^{q_{j'}}_{b,1}})\Bigg) \wedge \\ & \qquad t' > 0 \;\Rightarrow\; \{z_{j'}, c_{i'}\}\ \underline{\mathtt{in}}\ \Bigg((x_{i'} = y_{i'} \;\Rightarrow\; \psi^{i'}_{\theta^{q_{j'}}_{a,2}}) \wedge (x_{i'} < y_{i'} \;\Rightarrow\; \psi^{i'}_{\theta^{q_{j'}}_{b,2}})\Bigg)\Bigg]\Bigg)\Bigg]\end{aligned}$$

with

$$\psi^i_\theta \stackrel{\text{def}}{=} g^i_\alpha \wedge r^i_{\alpha'} \cup \{t\}\ \underline{\mathtt{in}}\ \forall\!\!\!\forall\Big(t > 1 \;\Rightarrow\; \{z_{j'}, c_{i'}\}\ \underline{\mathtt{in}}\ X\Big).$$

After any transition from a non-accepting "or" configuration of $\mathscr{M}$ (and leading to an "and" configuration), there exists a transition (among the two possible ones) which leads to a non-accepting "or" configuration. Choosing one a-transition in $\mathscr{M}$ is used to choose which transition (among the two enabled ones) in $\mathscr{M}$ has to be performed to follow the non-accepting computation. □

Proof (of Proposition 22). To prove that the specification complexity of $L_s^- + \langle a\rangle$ model checking is PSPACE-hard, we adapt the proof used to show the PSPACE-hardness of L_ν^- nil-model checking (Lemma 13, Appendix B) using the same idea employed in the previous proof. A QBF instance

$$\Phi = Q_1 p_1 \cdots Q_n p_n \cdot \varphi$$

is reduced to a model checking problem $\mathscr{A} \models \bar{\Phi}$ where $\mathscr{M}$ is the TA described in Fig. 3 and $\bar{\Phi}$ is the formula: $\{t, x_1, \ldots, x_n\}$ $\underline{\texttt{in}}$ Ψ with:

$$\Psi \overset{\text{def}}{=} \forall\!\!\!\forall(t=1 \Rightarrow \ldots$$

$$\ldots \forall\!\!\!\forall\Big(t=i \Rightarrow \left|\begin{array}{c} \langle a\rangle\forall\!\!\!\forall[b] \\ \forall\!\!\!\forall\big(t<i+1 \Rightarrow \end{array}\right| x_i\ \underline{\texttt{in}}\ \forall\!\!\!\forall(t=i+1 \Rightarrow \ldots\forall\!\!\!\forall(t=n+1 \Rightarrow \bar{\varphi}))\Big)\Big).$$

The subformula $\langle a\rangle\forall\!\!\!\forall[b]\ldots$ replaces $\exists\!\!\!\exists(t < i+1 \wedge \ldots)$, which was previously used for L_ν^-. As for Lemma 13, we have

$$\bar{\varphi} \overset{\text{def}}{=} \varphi[p_i \leftarrow x_i = n+1-i,\ \bar{p}_i \leftarrow x_i < n+1-i]_{i=1}^{n}. \qquad \square$$

Proof (of Proposition 23). Since $L_s + \vee$ is a sublanguage of L_ν, it is sufficient to argue that the (specification) complexity of the model checking problem for $L_s + \vee$ is EXPTIME-hard. To this end, note that the formula Φ (cf. Eqs. (B.1)–(B.4) in Appendix B) used to prove that the nil model checking problem for L_ν is EXPTIME-complete (Lemma 13) belongs to $L_s + \vee$. It follows that the model checking problem, and the nil model checking problem for $L_s + \vee$ are EXPTIME-complete. Since model checking for the nil process is EXPTIME-complete, so is the specification complexity of model checking.

Program complexity. We show that the program complexity of $L_s + \vee$ model checking is EXPTIME-hard. Let $\mathscr{M}$ be a LBATM and w_0 be an input word. As for the proof of Proposition 19, we assume without loss of generality that for each control state q, there are 2 transitions ($\theta^q_{a,1}$ and $\theta^q_{a,2}$) of the form $(q, a, \ldots)$ and two transitions ($\theta^q_{b,1}$ and $\theta^q_{b,2}$) of the form $(q, b, \ldots)$. We reduce the acceptance problem of w_0 by $\mathscr{M}$ to a model checking problem $A'_{\mathscr{M},w_0} \not\models \Phi$ where $A'_{\mathscr{M},w_0}$ is built in the same way as in the proof of Theorem 9 except we have to replace the $\langle a\rangle$ operators: for any "and" state q and any position i of the tape head, we label by s_ϵ the transition from (q, i) which simulates $\theta^q_{\alpha,\epsilon}$ for $\epsilon \in \{1, 2\}$. And we use the following formula for Φ:

$$\Phi \overset{\text{def}}{=} \forall\!\!\!\forall[s_0]\texttt{max}\Big(X, [\texttt{accept}]\texttt{ff} \wedge \forall\!\!\!\forall\ [s]\ \forall\!\!\!\forall\Big([s_1]X \vee [s_2]X\Big)\Big).$$

The idea is that after the initialization step, the configuration is not accepting and after an s-transition, any delay of duration d with $d \neq 1$ leads to a state from which no s_i is enabled (and then the subformula $[s_1]X \vee [s_2]X$ holds) and a delay of length 1 leads to a state from which there exists a transition (labeled by s_1 or s_2) leading to a non-accepting configuration. Note that only one s_1-transition and one s_2-transition are enabled. Clearly the previous formula does not depend of $\mathscr{M}$ and w_0. $\square$

Proof (of Proposition 25). (Specification) complexity. Since $L_s + \exists\!\!\exists$ is a sublanguage of L_ν, it is sufficient to argue that the (specification) complexity of the model checking problem for $L_s + \exists\!\!\exists$ is EXPTIME-hard. To this end, we begin by recalling that the formula Φ (cf. Eqs. (B.1)–(B.4) in Appendix B) used to prove that the *nil* model checking problem for L_ν is EXPTIME-complete (Lemma 13) belongs to $L_s + \vee$. Nevertheless it is possible to mimic the effect of general disjunction by means of $\exists\!\!\exists$. The problem consists in writing, without using $\vee$, the following subformula:

$$\{z_j, c_i\} \;\underline{\mathtt{in}}\; \bigvee_{\theta \in T(q_j)} \psi^i_\theta \tag{C.1}$$

of (B.2), where $T(q_j)$ is a set of transitions, and each formula ψ^i_θ is defined as in (B.3). Let $T(q_j)$ be $\{\theta_1, \ldots, \theta_m\}$ ($m \geqslant 0$). Note that the truth value of each formula ψ^i_θ is stable under delay transitions, if the clocks z_j and c_i are reset after the delay. This is because ψ^i_θ considers only differences between clocks, except for z_j and c_i which need to be less than 1.

Let t' be a fresh clock. In lieu of (C.1), we are going to use the following formula:

$$\begin{aligned} t' \;\underline{\mathtt{in}}\; \exists\!\!\exists\Big(0 \leqslant t' < 1 \;\Rightarrow\; & \{z_j, c_i\} \;\underline{\mathtt{in}}\; \psi^i_{\theta_1} \wedge \ldots \\ m-2 \leqslant t' < m-1 \;\Rightarrow\; & \{z_j, c_i\} \;\underline{\mathtt{in}}\; \psi^i_{\theta_{m-1}} \wedge \\ t' \geqslant m-1 \;\Rightarrow\; & \{z_j, c_i\} \;\underline{\mathtt{in}}\; \psi^i_{\theta_m}\Big). \end{aligned}$$

Again, $\mathscr{M}$ accepts w iff *nil* does not satisfy the resulting modified version of the formula Φ (cf. Eqs. (B.1)–(B.4)). Therefore the *nil* model checking problem for $L_s + \exists\!\!\exists$ is EXPTIME-complete, and so are the model checking problem and the specification complexity.

Program complexity. We show that the program complexity of $L_s + \exists\!\!\exists$ model checking is EXPTIME-hard. We adapt the previous proof to avoid the use of $\vee$ operator in Φ. First we place the guard $0 < t < 1$ in transitions (of $A'_{\mathscr{M},w_0}$) labeled by s_1 and the guard $t \geqslant 1$ in transitions labeled by s_2. In this timed automaton, the delay between two action transitions is just strictly greater than 0 (and not equal to 1 as before). The formula Φ' is the following:

$$\begin{aligned} \Phi = \forall\!\!\forall[s_0]\mathtt{max}\Big(X, [\mathtt{accept}]\mathtt{ff} \wedge \forall\!\!\forall\,[s]\, \Big(t' \;\underline{\mathtt{in}}\; \exists\!\!\exists\Big(t' > 0 \wedge \\ \exists\!\!\exists((t' < 1 \;\Rightarrow\; [s_1]X) \wedge (t' \geqslant 1 \;\Rightarrow\; [s_2]X))\Big)\Big)\Big). \end{aligned}$$

The existential quantification over delay transition allows us to choose between the two enabled transitions s_1 and s_2. $\square$

D. Complexity results for timed automata with restricted guards

The timed automata we considered in the main body of the paper may use clock constraints of the form $x - y \sim k$, where $\sim \in \{<, >, =\}$. This type of constraints was employed in the proof of Lemma 7 to model configurations of linear bounded Turing machines by means of states of TA. In this section we shall show that the working of an LBTM on an input w can be simulated by a TA even if we restrict the guards to be boolean combinations of clock constraints of the form $x \sim k$. We also briefly discuss the complexity of model checking problems when timed automata used to describe systems and formulae used to express properties can only use this type of restricted guards.

Let $\mathscr{M} = \langle Q, \Sigma, q_0, q_F, T\rangle$ be a non-deterministic LBTM. We assume, without loss of generality, that $\Sigma = \{a, b\}$. Let w_0 be an input word over Σ^*. From $\mathscr{M}$ and w_0 we are going to build, in polynomial time, a timed automaton $A_{\mathscr{M},w_0}$ with restricted guards such that w_0 is accepted by $\mathscr{M}$ iff a distinguished node is reachable in $A_{\mathscr{M},w_0}$.

Let $n = |w_0|$. The timed automaton $A_{\mathscr{M},w_0}$ is constructed as follows. The set of nodes of $A_{\mathscr{M},w_0}$ is

$$\{(q,i) \mid q \in Q, 1 \leqslant i \leqslant n\} \cup \{init, end\} \cup \{(i,\theta,l) \mid 1 \leqslant i \leqslant n, \theta \in T,\\ l \in \{1,\dots,i-1,i+1,\dots,n\}\}.$$

As in the encoding with full TA, a node (q,i) denotes the current state q of $\mathscr{M}$ and the current position i of the tape head. The role played by the nodes (i,θ,l) will be explained below. Again the tape content of the configurations of $\mathscr{M}$ is encoded by means of appropriate clock valuations. For each tape cell C_j $(1 \leqslant j \leqslant n)$, we have one clock x_j whose value encodes the content of C_j as follows: if C_j contains an a (resp. b), we have $x_j \leqslant 1$ (resp. $x_j > 1$). In the encoding that we are about to present, the transitions of $A_{\mathscr{M},w_0}$ which correspond to steps of *$\mathscr{M}$ occur precisely every 1 time unit.* This will be ensured by means of a clock t that measures the time elapsing between these transitions in $A_{\mathscr{M},w_0}$. Writing an a on cell j will consist in resetting x_j; therefore when the next transition simulating a step of $\mathscr{M}$ will be performed, the value of x_j will be precisely 1. Writing a b on cell j will consist in *not* resetting x_j; therefore when the next transition simulating a step of $\mathscr{M}$ will be performed, the value of x_j will be strictly greater than 1 (since at the previous step x_j was greater or equal to 1). Note that this encoding, unlike the one we employed in the proof of Lemma 7, is not preserved by delay transitions. This means that we will have to update every clock at every step. To this end, we will use the auxiliary states (i,θ,l), whose intuitive meaning is that "the current tape head position is i, the transition θ is being performed and the content of cell l is being updated".

The action set of $A_{\mathscr{M},w_0}$ is $\{s_0, s, \mathtt{upd}, \mathtt{accept}\}$. (The new action symbol upd will label transitions that update the current cell contents.) For every transition $\theta = (q,\alpha,\alpha',\delta,q')$ of machine $\mathscr{M}$, we add, for every possible position of the tape head $i \in \{1,\dots,n\}$, transitions $(q,i) \xrightarrow{g,s,r} (i,\theta,l_0)$ such that
(1) g is $t = 1 \wedge x_i = 1$ (resp. $t = 1 \wedge x_i > 1$) if $\alpha = a$ (resp. $\alpha = b$),
(2) $r = \{x_i, t\}$ (resp. $r = \{t\}$) if $\alpha' = a$ (resp. $\alpha' = b$), and
(3) $l_0 = 1$ (resp. $l_0 = 2$) if $i > 1$ (resp. $i = 1$).
In order to preserve the encoding of the contents of all the tape cells that were unaffected by the transition θ in $\mathscr{M}$, for each $i \in \{1,\dots,n\}$ and θ, we add a set of $2(n-1)$ transitions whose role is to update the value of the clocks x_j with $j \neq i$. This is achieved by means of a sequence of transitions of length $n-1$ with a duration 0. The "updating transitions" we add are the following:
(1) $(i,\theta,l) \xrightarrow{t=0\wedge x_l=1,\mathtt{upd},\{x_l\}} (i,\theta,l+1)$ and $(i,\theta,l) \xrightarrow{t=0\wedge x_l>1,\mathtt{upd},\emptyset} (i,\theta,l+1)$ if $l \in \{1,\dots, i-2, i+1,\dots,n-1\}$,
(2) $(i,\theta,i-1) \xrightarrow{t=0\wedge x_{i-1}=1,\mathtt{upd},\{x_{i-1}\}} (i,\theta,i+1)$ and $(i,\theta,i-1) \xrightarrow{t=0\wedge x_{i-1}>1,\mathtt{upd},\emptyset} (i,\theta, i+1)$ if $i < n$,

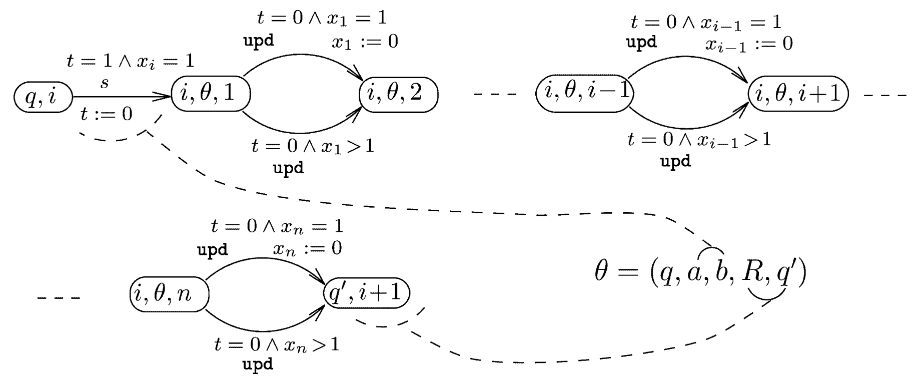

Fig. 4. The sequence of transitions to simulate the transition $\theta = (q, a, b, R, q')$.

(3) $(i,\theta,n) \overset{t=0\wedge x_n=1,\text{upd},\{x_n\}}{\longrightarrow} (q',i')$ and $(i,\theta,n) \overset{t=0\wedge x_n>1,\text{upd},\emptyset}{\longrightarrow} (q',i')$, where $i<n$, $\theta = (q,\alpha,\alpha',\delta,q')$ and $i' = i+1$ (resp. $i' = i-1$) if δ is R (resp. δ is L and $i>1$),

(4) $(n,\theta,n-1) \overset{t=0\wedge x_{n-1}=1,\text{upd},\{x_{n-1}\}}{\longrightarrow} (q',n-1)$ and $(i,\theta,n-1) \overset{t=0\wedge x_{n-1}>1,\text{upd},\emptyset}{\longrightarrow} (q',n-1)$ if $\theta = (q,\alpha,\alpha',\delta,q')$, and δ is L.

Fig. 4 shows an example simulation for a transition $\theta = (q,a,b,R,q')$ when $1<i<n$.

The initialization of the tape with the input word w_0 can be encoded by adding a transition $init \overset{t>1,s_0,r_{w_0}\cup\{t\}}{\longrightarrow} (q_0,1)$, where $r_{w_0} = \{x_i \mid w_0(i) = a\}$. To detect the accepting run, we add transitions $(q_F,i) \overset{\texttt{accept}}{\longrightarrow} end$ for every i. Clearly the node end is reachable in $A_{\mathscr{M},w_0}$ iff $\mathscr{M}$ accepts w_0.

The number of states of $A_{\mathscr{M},w_0}$ is in $\mathrm{O}(|Q|\cdot n + |T|\cdot n^2)$. The number of transitions is in $\mathrm{O}(|T|\cdot n^2)$.

Since $A_{\mathscr{M},w_0}$ can obviously be built in polynomial time from $\mathscr{M}$ and w_0, we have thus proven that the acceptance problem for LBTMs can be reduced in polynomial time to the reachability problem in timed automata with restricted guards. Thus:[5]

Proposition D.1. *The reachability problem in timed automata with restricted guards is PSPACE-complete.*

This result has already been shown in [5,31], but our encoding will allow us to obtain results about model checking problems for the property languages we have studied in the main body of the paper without using constraints of the form $x-y\bowtie k$ and systems modelled by means of timed automata with restricted guards. Our results on this topic for the property languages with fixed point operators are summarized[6] in Table 7. Sketches for the justifications for the claims may be found below:

A1 We offer a polynomial time reduction from the acceptance of an input w_0 by an LBATM $\mathscr{M}$ to a model checking problem $A_{\mathscr{M},w_0} \models \Phi$ where $A_{\mathscr{M},w_0}$ is the timed automaton with restricted guards built as above, and Φ is an L_ν formula used to encode

[5] The PSPACE-membership comes from the PSPACE-membership of reachability for the more general timed automata considered in the main body of the paper.

[6] Note that the complexity of the corresponding *nil* model checking problems remains open.

Table 7
Model checking w.r.t. restricted yimed automata: languages with fixed points

	L_ν	$L_s + \langle a \rangle$	$L_s + \vee$	$L_s + \exists\!\!\!\exists$	L_s
MC	EXPTIME-C See A1	EXPTIME-C See A2	EXPTIME-C See A3	EXPTIME-C See A4	PSPACE-C See A5

the behaviour of $\mathscr{M}$ on input w_0. We assume without loss of generality that we have a strict alternation of "or" and "and" states in $\mathscr{M}$; moreover we remind the reader that we assume that the initial and final states of $\mathscr{M}$ are "or" states.
The formula Φ describing the alternating behaviour of $\mathscr{M}$ on input w_0 is

$$\forall\!\!\!\forall[s_0]\mathtt{max}\Big(X, [\mathtt{accept}]\mathtt{ff} \wedge \forall\!\!\!\forall[s]\underbrace{[\mathtt{upd}]\cdots[\mathtt{upd}]}_{(n-1)\text{times}}\exists\!\!\!\exists\langle s\rangle\underbrace{\langle\mathtt{upd}\rangle\cdots\langle\mathtt{upd}\rangle}_{(n-1)\text{times}} X\Big).$$

Note that this formula is appropriate because after any s-transition there exists a sequence of $n-1$ upd-transitions. Now, $\mathscr{M}$ accepts w_0 iff $A_{\mathscr{M},w_0} \not\models \Phi$.

A2 The following $L_s + \langle a \rangle$ formula can replace the previous one to express the alternating behavior of an LBATM:

$$\forall\!\!\!\forall[s_0]\mathtt{max}\Bigg(X, [\mathtt{accept}]\mathtt{ff}\wedge$$
$$\forall\!\!\!\forall[s]\underbrace{[\mathtt{upd}]\cdots[\mathtt{upd}]}_{(n-1)\text{times}}\Big(z\ \underline{\mathtt{in}}\ \forall\!\!\!\forall(z=1 \Rightarrow \langle s\rangle\underbrace{\langle\mathtt{upd}\rangle\cdots\langle\mathtt{upd}\rangle}_{(n-1)\text{times}} X)\Big)\Bigg).$$

The sequence $\exists\!\!\!\exists\langle s\rangle\cdots$ can be replaced by $z\ \underline{\mathtt{in}}\ \forall\!\!\!\forall(z=1 \Rightarrow \langle s\rangle\cdots$ since the delay between two s-transitions is 1 time unit and time is not progressing during the updating steps.

A3 Given an LBTAM $\mathscr{M}$, we can assume (as for the proof of Proposition 19 in Appendix C) that for any control state q of $\mathscr{M}$, there are two transitions ($\theta^q_{a,1}$ and $\theta^q_{a,2}$) of the form $(q, a, \ldots)$ and two transitions ($\theta^q_{b,1}$ and $\theta^q_{b,2}$) of the form $(q, b, \ldots)$. Let w_0 an input word and $n = |w_0|$. We reduce the acceptance problem of w_0 by $\mathscr{M}$ to a model checking problem $A_{\mathscr{M},w_0} \models \Phi$ where $A_{\mathscr{M},w_0}$ is the timed automaton with restricted guards built as above except that the transitions of $A_{\mathscr{M},w_0}$ which correspond to a $\theta^q_{\alpha,j}$ transition of $\mathscr{M}$ ($\alpha \in \{a, b\}$ and $j \in \{1, 2\}$) with q being an "and" state are labeled by s_j. Then we can use the following $L_s + \vee$ formula:

$$\forall\!\!\!\forall[s_0]\mathtt{max}\Bigg(X, [\mathtt{accept}]\mathtt{ff} \wedge \forall\!\!\!\forall[s]\underbrace{[\mathtt{upd}]\cdots[\mathtt{upd}]}_{(n-1)\text{times}}\Big(z\ \underline{\mathtt{in}}\ \forall\!\!\!\forall(z=1 \Rightarrow$$
$$([s_1]\underbrace{[\mathtt{upd}]\cdots[\mathtt{upd}]}_{(n-1)\text{times}} X \vee [s_2]\underbrace{[\mathtt{upd}]\cdots[\mathtt{upd}]}_{(n-1)\text{times}} X))\Big)\Bigg).$$

This is based on the fact that for any q and i, even if there are two transitions $(q, i)\xrightarrow{s_1}$, exactly one of them is enabled for a given configuration and then the universal modality $[s_1]$ can be used.
Moreover, during the "updating phase", one and only one upd-transition is enabled and then $\langle\mathtt{upd}\rangle$ can be replaced by $[\mathtt{upd}]$.

Table 8
Model checking w.r.t. restricted timed automata: languages without fixed points

	$L_\nu{}^-$	$L_s^- + \langle a\rangle$	$L_s^- + \vee$	$L_s^- + \exists\!\!\exists$	L_s^-
MC	PSPACE-C	PSPACE-C	coNP-C	PSPACE-C	coNP-C
nil − MC	PSPACE-C	coNP-C	coNP-C	PSPACE-C	coNP-C

A4 The previous proof can be adapted to the $L_s + \exists\!\!\exists$ case. We use the same TA $A_{\mathscr{M},w_0}$ to simulate the behavior of $\mathscr{M}$ over w_0. Now we use the following formula:

$$\forall\!\!\forall[s_0]\mathtt{max}\Big(X, [\mathtt{accept}]\mathtt{ff} \wedge \forall\!\!\forall[s][\mathtt{upd}]\cdots[\mathtt{upd}]\Big[z\ \underline{\mathtt{in}}\ \exists\!\!\exists\Big(z < 1 \wedge$$
$$(z = 0 \Rightarrow \exists\!\!\exists(z = 1 \wedge [s_1][\mathtt{upd}]\cdots[\mathtt{upd}]X)) \wedge$$
$$(z > 0 \Rightarrow \exists\!\!\exists(z = 1 \wedge [s_2][\mathtt{upd}]\cdots[\mathtt{upd}]X))\Big)\Big]\Big).$$

A5 The reachability problem in timed automata with restricted guards can be reduced in linear time to an L_s model checking problem w.r.t. timed automata with restricted guards (as in Theorem 16).

Table 8 presents the complexities of model checking for the fixed point free fragments of the property languages we considered in this appendix. In this case, the proofs of the lower bounds of the corresponding problems for languages with extended guards use only restricted ones, and can be used here. Moreover the upper bounds are obtained directly from complexities of the full formalisms.

References

[1] L. Aceto, P. Bouyer, A. Burgueño, K.G. Larsen, The power of reachability testing for timed automata, in: Proceedings of FSTTCS'98, December, LNCS, vol. 1530, 1998, pp. 245–256.

[2] L. Aceto, A. Burgueño, K.G. Larsen, Model checking via reachability testing for timed automata, in: Proceedings of TACAS '98, LNCS, vol. 1384, April 1998, pp. 263–280.

[3] L. Aceto, A. Ingólfsdóttir, M.L. Pedersen, J. Poulsen, Characteristic formulae for timed automata, RAIRO, Theoretical Informatics and Applications 34 (2000) 565–584.

[4] L. Aceto, F. Laroussinie, Is your model checker on time? in: Proceedings of the 24th International Symposium on Math. Found. Comp. Sci. (MFCS'99), Szklarska Poreba, Poland, September 1999, Lecture Notes in Computer Science, vol. 1672, Springer, Berlin, 1999, pp. 125–136.

[5] R. Alur, Techniques for automatic verification of real-time systems, Ph.D. Thesis, Stanford University, 1991.

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

[7] R. Alur, D. Dill, A theory of timed automata, Theoretical Computer Science 126 (1994) 183–235.

[8] R. Alur, T. Feder, T.A. Henzinger, The benefits of relaxing punctuality, Journal of the ACM 43 (1996) 116–146.

[9] R. Alur, T.A. Henzinger, Logics and models of real time: a survey, in: J.W. de Bakker, K. Huizing, W.-P. de Roever, G. Rozenberg (Eds.), Proceedings of the REX Workshop 'Real-Time: Theory in Practice', Lecture Notes in Computer Science, vol. 600, Springer, Berlin, 1992, pp. 74–106.

[10] R. Alur, T.A. Henzinger, Real-time logics: complexity and expressiveness, Information and Computation 104 (1993) 35–77 (Preliminary version appears in the Proceedings of the 5th LICS, 1990).

[11] R. Alur, T.A. Henzinger, A really temporal logic, Journal of the ACM 41 (1994) 181–204 (Preliminary version appears in the Proceedings of the 30th FOCS, 1989).

[12] H.R. Andersen, Model checking and Boolean graphs, Theoretical Computer Science 126 (1994) 3–30.

[13] H.R. Andersen, Partial model checking (extended abstract), in: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science, LICS'95, La Jolla, San Diego, June 26–29, 1995, IEEE Computer Society Press, New York, 1995, pp. 398–407.

[14] A. Arnold, P. Crubille, A linear algorithm to solve fixed-point equations on transition systems, Information Processing Letters 29 (1988) 57–66.

[15] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, M. Yoel, Methodology and system for practical formal verification of reactive hardware, in: Ref. [34], pp. 182–193.

[16] G. Behrmann, K. Larsen, J. Pearson, C. Weise, W. Yi, Efficient timed reachability analysis using clock difference diagrams, in: Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99), Trento, Italy, July 1999, Lecture Notes in Computer Science, vol. 1633, Springer, Berlin, 1999.

[17] R. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.

[18] J. Bengtsson, D. Griffioen, K. Kristoffersen, K.G. Larsen, F. Larsson, P. Pettersson, W. Yi, Verification of an audio protocol with bus collision using Uppaal, in: R. Alur, T.A. Henzinger (Eds.), Proceedings of the 8th International Conference on Computer-Aided Verification, CAV'96, New Brunswick, NJ, USA, July 31–August 3, 1996, Lecture Notes in Computer Science, vol. 1102, Springer, Berlin, 1996.

[19] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, Systems and Software Verification, Model-Checking Techniques and Tools, Springer, Berlin, 2001.

[20] A. Biere, A. Cimatti, E.M. Clarke, Y. Zhu, Symbolic model checking without BDDs, in: W.R. Cleaveland (Ed.), Tools and Algorithms for the Construction and Analysis of Systems, TACAS'99, Amsterdam, The Netherlands, March 1999, Lecture Notes in Computer Science, vol. 1579, Springer, Berlin, 1999, pp. 193–207.

[21] R.E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Transactions on Computers C-35 (1986) 677–691.

[22] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer, Alternation, Journal of the ACM 28 (1981) 114–133.

[23] Z. Chaochen, C. Hoare, A. Ravn, A calculus of durations, Information Processing Letters 40 (1991) 269–276.

[24] E. Clarke, E. Emerson, Design and synthesis of synchronization skeletons using branching-time temporal logic, in: D. Kozen (Ed.), Proceedings of the Workshop on Logic of Programs Yorktown Heights, Lecture Notes in Computer Science, vol. 131, Springer, Berlin, 1981, pp. 52–71.

[25] E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite state concurrent system using temporal logic, ACM Transactions on Programming Languages and Systems 8 (1986) 244–263.

[26] E.M. Clarke, T. Filkorn, S. Jha, Exploiting symmetry in temporal logic model checking, in: Ref. [30], pp. 450–462.

[27] E.M. Clarke, O. Grumberg, D.A. Peled, Model Checking, MIT Press, Cambridge, MA, 1999.

[28] E.M. Clarke, J.M. Wing, Formal methods: State of the art and future directions, ACM Computing Surveys 28 (1996) 626–643 (Report by the Working Group on Formal Methods for the ACM Workshop on Strategic Directions in Computing Research).

[29] R. Cleaveland, A linear-time model-checking algorithm for the alternation-free modal μ-calculus, Formal Methods in Systems Design 2 (1993) 121–147.

[30] C. Courcoubetis (Ed.), Proceedings of the 5th International Workshop in Computer Aided Verification, CAV'93, Elounda, Greece, June/July 1993, Lecture Notes in Computer Science, vol. 697, Springer, Berlin, 1993.

[31] C. Courcoubetis, M. Yannakakis, Minimum and maximum delay problems in real-time systems, Formal Methods in System Design (1992) 385–415.

[32] S. Demri, P. Schnoebelen, The complexity of propositional linear temporal logics in simple cases (extended abstract), in: Proceedings of the 15th Annual Symposium Theoretical Aspects of Computer Science (STACS'98), Paris, France, February 1998, LNCS, vol. 1373, Springer, Berlin, 1998, pp. 61–72.

[33] D. Dill, Timing assumptions and verification of finite-state concurrent systems, in: Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, LNCS, vol. 407, 1989.

[34] D. Dill (Ed.), Proceedings of the International Conference on Computer-Aided Verification, CAV'94, Stanford, CA, USA, June 1994, Lecture Notes in Computer Science, vol. 818, Springer, Berlin, 1994.

[35] D. Drusinsky, D. Harel, On the power of bounded concurrency I: Finite automata, Journal of the ACM 41 (1994) 517–539.

[36] S. Dziembowski, M. Jurdziński, D. Niwiński, On the expression complexity of the modal μ-calculus model checking, unpublished manuscript, November 1996.

[37] E. A. Emerson, C.S. Jutla, A.P. Sistla, On model-checking for fragments of μ-calculus, in; Ref. [30], pp. 385–396.

[38] E.A. Emerson, C.-L. Lei, Efficient model checking in fragments of the propositional mu-calculus, in: Proceedings of the 1st Annual Symposium on Logic in Computer Science, LICS'86, IEEE Computer Society Press, Silver Spring, MD, 1986, pp. 267–278.

[39] D. Harel, O. Kupferman, M.Y. Vardi, On the complexity of verifying concurrent transition systems, in: Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97), Warsaw, Poland, July 1997, Lecture Notes in Computer Science, vol. 1243, Springer, Berlin, 1997, pp. 258–272.

[40] E. Harel, O. Lichtenstein, A. Pnueli, Explicit clock temporal logic, in: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, Philadelphia, Pennsylvania, 4–7 June 1990, IEEE Computer Society Press, Silver Spring, MD, 1985, pp. 402–413.

[41] M. Hennessy, R. Milner, Algebraic laws for nondeterminism and concurrency, Journal of the ACM 32 (1985) 137–161.

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

[43] Y. Hirshfeld, A. Rabinovich, A framework for decidable metrical logics, in: Proceedings of the 26th International Colloqium on Automata, Languages, and Programming (ICALP'99), Prague, Czech Republic, July 1999, Lecture Notes in Computer Science, vol. 1644, Springer, Berlin, 1999, pp. 422–432.

[44] Y. Hirshfeld, A. Rabinovich, Quantitative temporal logic, in: Proceedings of the 13th International Workshop on Computer Science Logic (CSL'99), Madrid, Spain, September 1999, Lecture Notes in Computer Science, vol. 1683, Springer, Berlin, 1999, pp. 172–187.

[45] P.-H. Ho, H. Wong-Toi, Automated analysis of an audio control protocol, in: P. Wolper (Ed.), Proceedings of the 7th International Conference on Computer-Aided Verification, CAV'95, Liège, Belgium, July 1995, Lecture Notes in Computer Science, vol. 939, Springer, Berlin, 1995, pp. 381–394.

[46] H. Hüttel, K.G. Larsen, The use of static constructs in a modal process logic, Lecture Notes in Computer Science, Springer, Berlin, 1989.

[47] M. Jurdziński, Deciding the winner in parity games is in UP $\cap$ co-UP, Information Processing Letters 68 (1998) 119–124.

[48] R.M. Karp, Reducibility among combinatorial problems, Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–104.

[49] D. Kozen, Lower bounds for natural proof systems, in: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, IEEE, 1977, pp. 254–266.

[50] D. Kozen, Results on the propositional μ-calculus, Theoretical Computer Science 27 (1983) 333–354.

[51] O. Kupferman, M.Y. Vardi, P. Wolper, An automata-theoretic approach to branching-time model checking, Journal of ACM 47 (2000) 312–360.

[52] F. Laroussinie, K.G. Larsen, Compositional model checking of real time systems, in: I. Lee, S. Smolka (Eds.), Proceedings of the 6th International Conference on Concurrency Theory, CONCUR'95, Philadelphia, PA, USA, August 21–24, 1995, Lecture Notes in Computer Science, vol. 962, Springer, Berlin, 1995.

[53] F. Laroussinie, K.G. Larsen, CMC: A tool for compositional model-checking of real-time systems, in: Proceedings of the IFIP Joint International Conference on Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98), Kluwer Academic Publishers, Dordrecht, 1998, pp. 439–456.

[54] F. Laroussinie, K.G. Larsen, C. Weise, From timed automata to logic - and back, in: J. Wiedermann, P. Hájek (Eds.), Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science, MFCS'95, Prague, Czech Republic, August 28–September 1, 1995, Lecture Notes in Computer Science, vol. 969, Springer, Berlin , 1995, pp. 529–539.

[55] F. Laroussinie, N. Markey, Ph. Schnoebelen, Model checking CTL^+ and $FCTL$ is hard, in: Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'2001), Genova, Italy, April 2001, Lecture Notes in Computer Science, vol. 2030, Springer, 2001, pp. 318–331.

[56] F. Laroussinie, Ph. Schnoebelen, The state explosion problem from trace to bisimulation equivalence, in: Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'2000), Berlin, Germany, March–April 2000, Lecture Notes in Computer Science, vol. 1784, Springer, 2000, pp. 192–207.

[57] K.G. Larsen, P. Pettersson, W. Yi, Model-checking for real-time systems, in: H.R. (Ed.), Proceedings of the 10th International Conference on Fundamentals of Computation Theory, Dresden, Germany, August 1995, LNCS, vol. 965, pp. 62–88.

[58] K.G. Larsen, P. Pettersson, W. Yi, UPPAAL in a Nutshell, Journal of Software Tools for Technology Transfer 1 (1997) 134–152.

[59] O. Lichtenstein, 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, January 1985, pp. 97–107.

[60] O. Maler, S. Yovine, Hardware timing verification using KRONOS, in: Proceedings of the 7th Israeli Conference on Computer Systems and Software Engineering, Herzliya, Israel, June, 1996.

[61] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, MA, 1994.

[62] J.P. Queille, J. Sifakis, Specification and verification of concurrent systems, in Cesar, in: Proceedings of the 5th International Symposium on Programming, Lecture Notes in Computer Science, vol. 137, Springer, Berlin, 1981, pp. 337–357.

[63] A. Rabinovich, Complexity of equivalence problems for concurrent systems of finite agents, Information and Computation 139 (1997) 111–129.

[64] A. Rabinovich, Symbolic model checking for μ-calculus requires exponential time, Theoretical Computer Science 243 (2000) 467–475.

[65] Y.S. Ramakrishna, P.M. Melliar-Smith, L.E. Moser, L.K. Dillon, G. Kutty, Interval logics and their decision procedures. Part II: A real-time interval logic, Theoretical Computer Science 170 (1996) 1–46 (Fundamental study).

[66] W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities, Journal of the Computer and System Sciences 4 (1970) 177–192.

[67] A.P. Sistla, E.M. Clarke, The complexity of propositional linear temporal logics, Journal of the ACM 32 (1985) 733–749.

[68] L.J. Stockmeyer, The complexity of decision problems in automata theory and logic, Technical Report MAC TR-133, Project MAC, MIT Press, Cambridge, MA, 1974.

[69] L.J. Stockmeyer, A.R. Meyer, Word problems requiring exponential time: Preliminary report, in: Conference Record of Fifth Annual ACM Symposium on Theory of Computing, Austin, Texas, 30 April–2 May 1973, pp. 1–9.

[70] T.A. Henzinger, P.-H. Ho, H. Wong-Toi, HyTech: A model checker for hybrid systems, Journal of Software Tools for Technology Transfer 1 (1997) 110–122.

[71] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pacific Journal of Mathematics 5 (1955).

[72] M.Y. Vardi, Linear vs. branching time: A complexity-theoretic perspective, in: V. Pratt (Ed.), Proceedings of the 13th Annual Symposium on Logic in Computer Science, LICS'98, IEEE Computer Society Press, Silver Spring, MD, 1998.

[73] M.Y. Vardi, P. Wolper, Reasoning about infinite computations, Information and Computation 115 (1994) 1–37.

[74] Th. Wilke, Specifying timed state sequences in powerful decidable logics and timed automata, in: H. Langmaack, W.-P. de Roever, J. Vytopil (Eds.), Formal Techniques in Real-Time and Fault-Tolerant Systems, Lecture Notes Computer Science, Lübeck, vol. 863, Springer, Berlin, 1994, pp. 694–715.

[75] M. Yannakakis, D. Lee, An efficient algorithm for minimizing real-time transition systems, in: Ref. [30], pp. 210–224.

[76] S. Yovine, Kronos: A verification tool for real-time systems, Journal of Software Tools for Technology Transfer 1 (1997) 123–133.