On the Complexity of LTL Model-Checking of Recursive State Machines*

Salvatore La Torre[1] and Gennaro Parlato[1,2]

[1]Università degli Studi di Salerno, Italy
[2]University of Illinois at Urbana-Champaign, USA

Abstract. Recursive state machines (RSMs) are models for programs with recursive procedural calls. While LTL model-checking is EXPTIME-complete on such models, on finite-state machines, it is PSPACE-complete in general and becomes NP-complete for interesting fragments. In this paper, we systematically study the computational complexity of model-checking RSMs against several syntactic fragments of LTL. Our main result shows that if in the specification we disallow *next* and *until*, and retain only the *box* and *diamond* operators, model-checking is in NP. Thus, differently from the full logic, for this fragment the abstract complexity of model-checking does not change moving from finite-state machines to RSMs. Our results on the other studied fragments confirm this trend, in the sense that, moving from finite-state machines to RSMs, the complexity of model-checking either rises from PSPACE-complete to EXPTIME-complete, or stays within NP.

1 Introduction

Linear temporal logic (LTL) is a specification language commonly used for expressing correctness requirements of reactive systems [10,11]. An LTL formula is built from atomic propositions, boolean connectives, and temporal modalities such as *next*, *diamond*, *box*, and *until*, and is interpreted over infinite sequences of states which model computations of reactive programs. A typical temporal requirement is "every request p is followed by a response q," and can be expressed in LTL as $\Box\ (\ p \rightarrow \Diamond\ q\ )$.

Given an abstract model M of a reactive system and an LTL formula φ, LTL *model-checking* asks whether an infinite computation of M satisfying φ exists. When M is finite-state, this decision problem is PSPACE-complete [12]. Despite the high computational complexity, solutions to LTL model-checking are implemented in verification tools and successfully applied in practice (see [5,9]).

The control flow of programs with recursive procedure calls can be naturally captured with *recursive state machines* (RSMs). In an RSM, vertices can either be ordinary states or can correspond to invocations of other state machines in a potentially recursive manner. We recall that RSMs correspond to pushdown

* This research was partially supported by the MIUR grant ex-60% 2005, Università degli Studi di Salerno.

L. Arge et al. (Eds.): ICALP 2007, LNCS 4596, pp. 937–948, 2007.

systems, and the related model-checking problem has been extensively studied in the literature [1,4,8,13]. Among the many interesting results, we recall that LTL model-checking of RSMs is EXPTIME-complete [4].

On finite-state systems, the computational complexity of model-checking for meaningful LTL fragments improves (see [6]). In particular, for the logic of all formulae obtained by allowing an arbitrary nesting of *diamond* and *box* operators, which we denote with $\mathsf{L}(\Diamond)$, model-checking is NP-complete [12]. This fragment is very rich and can express many interesting properties of reactive systems, such as liveness (as the above response property) but also safety, fairness, and others.

In this paper, we systematically study the computational complexity of model-checking RSMs with respect to specifications from natural syntactic fragments of LTL. Our goal is the discovery of more efficient algorithms for interesting classes of specifications and a better understanding of the computational complexity of this decision problem. The main result of this paper concerns the computational complexity of model-checking RSMs with respect to $\mathsf{L}(\Diamond)$ specifications. We show that this problem is in NP, that matches the known lower bound for finite-state machines. Thus, differently from full LTL, the computational complexity of $\mathsf{L}(\Diamond)$ model-checking stays unchanged while moving from finite-state systems to RSMs. Considering the simple structure of $\mathsf{L}(\Diamond)$ models [12], this result may not seem surprising. However, the effects of mixing this logic with the use of a stack were not completely clear. In the EXPTIME-hardness proof for LTL [4], the stack of the model is used to linearize computations of an alternating Turing machine working in polynomial space and this basically allows the authors to carry over the PSPACE-hardness proof of the LTL model-checking of finite-state machines. Moreover, the expressiveness of $\mathsf{L}(\Diamond)$ combined with the alternation provided by a 2-player game graph suffices to prove a matching lower bound for deciding full LTL games [3].

To show NP membership for the model-checking of RSMs with respect to $\mathsf{L}(\Diamond)$ specifications, we define a certificate (a finite sequence of alternate nodes and set of nodes of the RSM) and then show that the problem can be reduced to answering two distinct queries: (1) checking that the certificate satisfies the given formula, and (2) checking that there exists a run of the given RSM which matches the certificate. To complete the NP membership argument we also show that we only need to investigate certificates of length bounded by the size of the formula. Observe that the smallest model of an $\mathsf{L}(\Diamond)$ formula among all the runs of an RSM can be of size over-polynomial, thus looking for an abstract representation of runs is needed and clearly the arguments used in showing NP membership of this problem on finite Kripke structures [12] cannot be applied here.

The other results of this paper concern the model-checking of RSMs with respect to specifications expressed in other syntactic fragments of LTL. For fragments whose model-checking problem on finite-state machines is PSPACE-complete, we show EXPTIME-hardness, and therefore the model-checking problem is EXPTIME-complete. For fragments whose model-checking problem on finite-state machines is NP-complete, we instead give an NP upper bound. Remarkably, for some of these fragments we re-use results shown to prove NP

membership for $\mathsf{L}(\Diamond)$. All the results are summarized in a table in the concluding section.

2 Logic and Models

2.1 Propositional Linear Temporal Logic

In this section, we briefly recall the logic LTL and LTL model-checking (see [7]).

Given a set of atomic propositions AP, a *propositional linear temporal logic* (LTL) formula is composed of atomic propositions from AP, the boolean connectives *negation* ($\neg$), *conjunction* ($\wedge$) and *disjunction* ($\vee$), the temporal operators *next* ($\bigcirc$), *diamond* ($\Diamond$), *box* ($\Box$), and *until* ($\mathcal{U}$). Formulae are built up in the usual way from these operators and connectives, according to the following grammar

$$\varphi := p \,|\, \neg\varphi \,|\, \varphi \wedge \varphi \,|\, \bigcirc\varphi \,|\, \varphi \,\mathcal{U}\, \varphi,$$

where p is an atomic proposition of AP. The other formulae can be introduced as abbreviations in the usual way. In particular, the diamond operator is a restricted form of until (i.e., $\Diamond\varphi \equiv \text{TRUE}\,\mathcal{U}\,\varphi$ holds), and the box operator expresses the logical negation of the diamond operator (i.e., $\Diamond\varphi \equiv \neg(\Box\neg\varphi)$ holds).

LTL formulae are interpreted on linear-time structures. A *linear-time structure* (or simply a *structure*) is a pair (π, lab_π) where π is an ω-sequence $\pi = s_0 s_1 s_2 \ldots$ of *states* and lab_π is a mapping $lab_\pi : \{s_0, s_1, s_2, \ldots\} \to 2^{AP}$ labeling each state s_i with the set of propositions that hold at s_i.

Let π be a structure, $i \in \mathbb{N}_0$ be a position, and φ an LTL formula. The satisfaction relation $\models$ is inductively defined as follows.

- $(\pi, i) \models p \overset{\text{def}}{\Leftrightarrow} p \in lab_\pi(s_i)$;
- $(\pi, i) \models \neg\varphi \overset{\text{def}}{\Leftrightarrow} (\pi, i) \models \varphi$ holds false (i.e., $(\pi, i) \not\models \varphi$);
- $(\pi, i) \models \varphi \wedge \psi \overset{\text{def}}{\Leftrightarrow} (\pi, i) \models \varphi$ and $(\pi, i) \models \psi$;
- $(\pi, i) \models \bigcirc\varphi \overset{\text{def}}{\Leftrightarrow} (\pi, i+1) \models \varphi$;
- $(\pi, i) \models \varphi \;\mathcal{U}\; \psi \overset{\text{def}}{\Leftrightarrow} \exists j \geq i$ such that $(\pi, j) \models \psi$, and $\forall k \in [i, j-1], (\pi, k) \models \varphi$;

When $(\pi, 0) \models \varphi$, we also write $\pi \models \varphi$ and say that π is a *model* of φ.

An LTL-formula φ is *satisfiable* if there exists a model of φ. Given an LTL formula φ, we denote with $|\varphi|$ its length. We denote with $\mathsf{L}(\Diamond)$ the LTL fragment that allows only $\Diamond$ and $\Box$ as temporal operators (no next or until operators are allowed).

Given a set of atomic propositions AP, a *Kripke structure* over AP is a tuple $M = (S, \Delta, lab, I)$ where: S is a (possibly infinite) set of *states*; $\Delta \subseteq S \times S$ is a *transition relation*; $lab : S \to 2^{AP}$ is a *labeling function* that associates with each state s a set of atomic propositions $lab(s)$ (meaning that the atomic propositions that hold true at s are exactly those in $lab(s)$); $I \subseteq S$ is the set of the initial states of M.

We say that M is a *finite* Kripke structure when S is finite. A *run* or *path* of M is a (finite or infinite) sequence of states $\pi = s_0 s_1 \ldots$ of M, such that $s_0 \in I$

and (s_i, s_{i+1}) is a transition of M, with $i \geq 0$. Notice that, each run in M is a linear-time structure. We write $M \models \varphi$ when there exists an infinite run π of M such that $\pi \models \varphi$. The *model-checking* problem is: "given a Kripke structure M and an LTL formula φ, does $M \models \varphi$?" With $|M|$ we denote the size of M, that is $|S| + |\Delta|$.

In the following, we use the result.

Theorem 1 ([12]). *The model-checking problem of a finite Kripke structure against an $L(\Diamond)$ formula is* NP*-complete.*

2.2 Recursive State Machines

In this section, we recall the recursive state machines (RSMs) introduced in [1].

Syntax. Let $I_{\mathcal{M}} = \{1, \ldots, k\}$. A *recursive state machine* (RSM) $\mathcal{M}$ over a set of atomic propositions AP is a tuple $(M_1, \ldots, M_k)$, where $M_h = (N_h, B_h, Y_h, En_h, Ex_h, E_h, true_h)$, for $h \in I_{\mathcal{M}}$, is a module and consists of the following components:

- A finite nonempty set of *nodes* N_h, and a finite set of *boxes* B_h.
- A labeling $Y_h : B_h \to I_{\mathcal{M}}$ that assigns to every box a module index.
- A set of *entry* nodes $En_h \subseteq N_h$, and a set of *exit* nodes $Ex_h \subseteq N_h$.
- Let $Calls_h = \{(b, e) | b \in B_h, e \in En_j, j = Y_h(b)\}$ denote the set of *calls* of M_h, and $Retns_h = \{(b, x) | b \in B_h, x \in Ex_j, j = Y_h(b)\}$ denote the set of *returns* in M_h. Then, $E_h \subseteq ((N_h \cup Retns_h) \times (N_h \cup Calls_h))$ is the set of *edges* of M_h.
- A *labeling function* $true_h : N_h \to 2^{AP}$ that associates a set of atomic propositions to each node of M_h.

We assume that $N_1, \ldots, N_k$ (respectively, $B_1, \ldots, B_k$) are pairwise disjoint. We define with $N = \bigcup_{i \in I_{\mathcal{M}}} N_i$ the set of all *nodes* of $\mathcal{M}$, with $B = \bigcup_{i \in I_{\mathcal{M}}} B_i$ the set of all *boxes* of $\mathcal{M}$, and with $E = \bigcup_{i \in I_{\mathcal{M}}} E_i$ the set of all *edges* of $\mathcal{M}$. Also, $Y : B \to I_{\mathcal{M}}$ denotes the extension of all the functions $Y_h, h \in I_{\mathcal{M}}$, and $true : N \to 2^{AP}$ denotes the extension of all the functions $true_h$ $(h \in I_{\mathcal{M}})$. Module M_k is the *initial* module of $\mathcal{M}$.

Fig. 1 illustrates the definition. The RSM has two modules. The module M_1 has 5 nodes, of which e_1 and e_2 are the entry nodes and x_1 and x_2 are the exit nodes, and two boxes, of which b_1 is mapped to module M_2 and b_2 is mapped to M_1. The entry and exit nodes are the control interface of a module by which it can communicate with the other modules. Intuitively, think of modules as procedures, and an edge entering a box invoking the procedure associated with the box.

Semantics. With each module of an RSM, we associate a Kripke structure by recursively substituting each box by the module referenced by the box. Formally, we associate to the module M_h the Kripke structure $M_h^F = (S_h, \Delta_h, lab_h, I_h)$, called the *expansion* or the *flat* of M_h in $\mathcal{M}$, as follows. The states S_h of M_h^F are elements of the form $\langle \alpha u \rangle$ where $\alpha u \in B^* \times N$ and either

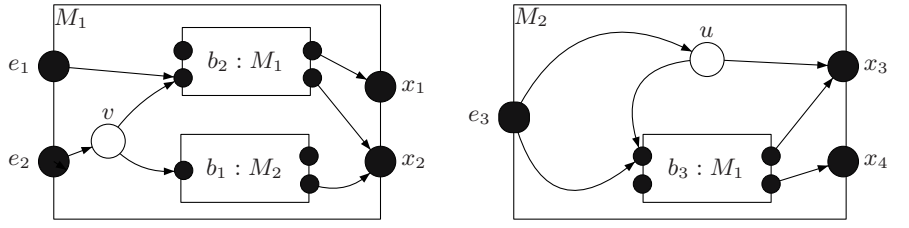

Fig. 1. A sample recursive state machine

- $\alpha = \epsilon$ and $u \in N_h$, or,
- $\alpha = b_1 \dots b_\ell$ (with $\ell \geq 1$), $b_1 \in B_h$, $b_{i+1} \in B_{Y(b_i)}$ for every $i \in [1, \ell - 1]$, and $u \in N_{Y(b_\ell)}$.

The initial states of M_h^F are $I_h = \{\langle e \rangle | e \in En_h\}$. The labeling function is $lab_h(\langle \alpha u \rangle) = true(u)$. In the following, for the ease of presentation, we also denote the states of $\mathcal{M}$ of the form $\langle \alpha b u \rangle$, where (b, u) is a call or a return, as $\langle \alpha(b, u) \rangle$. Therefore, states corresponding to calls or returns will have two representations and of the two we will always use the more convenient one, as for the following definition. Given two states X, X' of M_h^F, we define the transition relation Δ_h as: $(X, X') \in \Delta_h$ iff $X = \langle \alpha u \rangle$, $X' = \langle \alpha v \rangle$, and (u, v) is an edge of $\mathcal{M}$. (Note that in order to make the above definition consistent, we implicitly use the representation of the form $\langle \beta(b, z) \rangle$ for X, if (b, z) is a return, and for X', if (b, z) is a call, and the standard representation in all the other cases.) The expansion M_k^F of the initial module of $\mathcal{M}$ is also denoted $\mathcal{M}^F$.

Given a state $X = \langle \alpha u \rangle$ with $u \in N$, we denote with $node(X)$ the node u, and say that u is the *node* of X.

An RSM $\mathcal{M}$ satisfies an LTL formula φ, written $\mathcal{M} \models \varphi$, iff $\mathcal{M}^F \models \varphi$. The model-checking problem of an RSM $\mathcal{M}$ against an LTL formula φ is to determine whether $\mathcal{M}$ satisfies φ. The following result holds.

Theorem 2 ([1]). *The problem of model-checking an* RSM *against an* LTL *formula is* EXPTIME*-complete.*

3 The Complexity of $\mathsf{L}(\Diamond)$ Model-Checking on RSMs

In this section we show that model-checking of $\mathsf{L}(\Diamond)$ formulae against RSMs is NP-complete. We prove first some preliminary results and then use them to argue the claimed result.

Fix an RSM $\mathcal{M}$ and an $\mathsf{L}(\Diamond)$ formula φ. Our first goal is to define a finite certificate for a run that will be used in the argument to show membership in NP. An $\mathcal{M}$*-sequence* of length ℓ is a sequence $\gamma = u_0, U_1, u_2, U_3, \dots, u_{2\ell}, U_{2\ell+1}$ suc h that $u_0, u_2, \dots, u_{2\ell}$ are $\mathcal{M}$ nodes, and $U_1, U_3, \dots, U_{2\ell+1}$ are sets of $\mathcal{M}$ nodes. For a run π of $\mathcal{M}^F$ and an $\mathcal{M}$*-sequence* γ as above, we say that π and γ *match* if π can be factored as $X_0\pi_1X_2\pi_3 \dots X_{2\ell}\pi_{2\ell+1}$ where X_{2i} are states, π_{2i+1} are

runs, $node(X_{2i}) = u_{2i}$, and all the nodes of states occurring in π_{2i+1} belong to U_{2i+1}, for every $i \in [0, \ell]$, and for all $u \in U_{2\ell+1}$, u is the node of infinitely many states in $\pi_{2\ell+1}$.

It is crucial for our argument to be able to decide in polynomial time whether there exists an $\mathcal{M}^F$ run that matches a given $\mathcal{M}$-sequence γ. We can construct a simple Büchi automaton A of size linear in the sizes of $\mathcal{M}$ and γ which accepts infinite sequences matching γ. Therefore, the above problem reduces to checking for emptiness the intersection of $\mathcal{M}$ and A. Thus, by the results shown in [1,8] we have:

Lemma 1. *Given an $\mathcal{M}$-sequence γ, deciding if there exists a run of $\mathcal{M}^F$ matching γ can be done in deterministic polynomial time in the size of $\mathcal{M}$ and the length of γ.*

The second important step in our argument for showing NP membership consists of defining a notion of satisfiability of formulae on $\mathcal{M}$-sequences that can be checked efficiently and that is sufficient to show satisfiability on a matching run of $\mathcal{M}^F$.

Fix an $\mathcal{M}$-*sequence* $\gamma = u_0, U_1, u_2, U_3, \ldots, u_{2\ell}, U_{2\ell+1}$. We denote with $BD(\varphi)$ the set of all the box/diamond sub-formulae ψ of φ, and for a linear structure $\pi = s_0 s_1 s_2 \ldots$, we denote with $BD^{\varphi}_{\pi}(s_i)$ the set of all formulae ψ of $BD(\varphi)$ such that $(\pi, i) \models \psi$.

We say that γ *satisfies* φ if:

1. the sequence $\pi = u_0 u_2 \ldots u_{2\ell} \pi'$ is a model of φ, where π' is any infinite sequence over $U_{2\ell+1}$ such that each $u \in U_{2\ell+1}$ occurs as the node of infinitely many states of π';
2. for every $i \in [1, \ell]$ and for every u in U_{2i-1}, $BD^{\varphi}_{\hat{\pi}}(u) = BD^{\varphi}_{\pi}(u_{2i})$ where $\hat{\pi} = u u_{2i} \ldots u_{2\ell} \pi'$.

The following proposition elaborates on some results shown in [12] for the logic we are considering.

Proposition 1. *Let $\pi = s_0 s_1 s_2 \ldots$ be a structure and π' be the suffix of π such that for each state s in π' there are infinitely many states s' in π' such that $lab_{\pi}(s) = lab_{\pi}(s')$.*[1]

1. *For an $\mathsf{L}(\Diamond)$ formula φ, the set of φ sub-formulae that hold at a state s_i depends only on the atomic propositions that hold at s_i and the box/diamond sub-formulae that hold at s_{i+1} along π.*
2. *For $\varphi \in \mathsf{L}(\Diamond)$, $BD^{\varphi}_{\pi}(s') = BD^{\varphi}_{\pi}(s'')$ for every pair of states s', s'' within π'.*

The following lemma states that satisfaction of a formula φ on a given $\mathcal{M}$-sequence can be efficiently verified.

Lemma 2. *Given an $\mathsf{L}(\Diamond)$ formula φ and an $\mathcal{M}$-sequence γ, checking that γ satisfies φ can be done in polynomial time.*

[1] Note that such a suffix of π always exists since 2^{AP} is finite.

Proof. Checking part 1 of the definition of satisfiability on $\mathcal{M}$-sequences is trivial and can be done in linear time (see [12]). From Proposition 1, we can start computing $BD^{\varphi}_{\pi}(u)$ for a u in π', and then we compute the subsets $BD^{\varphi}_{\pi}(u_{2i})$ for $i = \ell, \ldots, 0$. Therefore, the lemma is proved. □

A crucial step in our argument for showing membership in NP is to prove that we can answer to our model-checking question simply searching for an $\mathcal{M}$-sequence γ which satisfies the given formula and checking for the existence of a run of $\mathcal{M}^F$ matching γ. The next two lemmas prepare such a result.

Lemma 3. *If an* $\mathcal{M}$-sequence γ *satisfies* φ *then each run of* $\mathcal{M}^F$ *which matches* γ *also satisfies* φ*.*

Proof. Let $\gamma = u_0, U_1, u_2, \ldots, U_{2\ell-1}, u_{2\ell}, U_{2\ell+1}$ be an $\mathcal{M}$ sequence satisfying φ and π be a run of $\mathcal{M}^F$ that matches γ. Thus, π can be factorized as $X_0\pi_1X_2\pi_3\ldots X_{2\ell}\pi_{2\ell+1}$ where for all $u \in U_{2\ell+1}$, u occurs infinitely often in $\pi_{2\ell+1}$ and for $i \in [0, \ell]$: X_{2i} are states, π_{2i+1} are runs, $node(X_{2i}) = u_{2i}$, and all the nodes of π_{2i+1} states belong to U_{2i+1}. By part 2 of Proposition 1, we have that we can compute the set $BD^{\varphi}_{\pi}(X)$ for each state X of $\pi_{2\ell+1}$ by simply computing it for one of them. Since the sets of atomic propositions which occur infinitely often in $\pi_{2\ell+1}$ are exactly the sets of atomic propositions which label the nodes of $U_{2\ell+1}$, by part 1 of Proposition 1, $BD^{\varphi}_{\pi}(u_0) = BD^{\varphi}_{\pi}(X_0)$ holds. Therefore, π satisfies φ and the lemma is proved. □

Lemma 4. *If a run* π *of* $\mathcal{M}^F$ *satisfies* φ *then there is an* $\mathcal{M}$-sequence *of length* $\ell \leq |BD(\varphi)| + 1$ *which satisfies* φ *and matches* π*.*

Proof. Let $\pi = X_0X_1X_2\ldots$ be a run of $\mathcal{M}^F$ that satisfies φ. Denote with ρ the sequence $s_0s_1s_2\ldots$ where $s_i = node(X_i)$ for all $i = 0, 1, 2\ldots$. We construct a matching $\mathcal{M}$-sequence γ as follows. For each diamond sub-formula ψ of φ, consider the maximum index i such that $(\rho, i) \models \psi$ (if defined). Instead, for each box sub-formula ψ of φ, consider the maximum index i such that $(\rho, i) \not\models \psi$ (if defined). Take the nodes corresponding to such indices and denote them as $u_2, \ldots, u_{2\ell-2}$ (by increasing indices). Observe that $\ell \leq |BD(\varphi)| + 1$. Since the number of $\mathcal{M}$ nodes is finite, there is a suffix ρ' of ρ starting after $u_{2\ell-2}$ such that all nodes of ρ' occur infinitely often in ρ'. Take as node $u_{2\ell}$ an arbitrary node in ρ' and as u_0 the first node of ρ. Therefore, ρ can be factorized as $u_0\rho_1u_2\ldots\rho_{2\ell-1}u_{2\ell}\rho_{2\ell+1}$. (Note that $\rho_{2\ell+1}$ is a suffix of ρ'.) Thus, we define γ as $u_0, U_1, u_2, \ldots, U_{2\ell-1}, u_{2\ell}, U_{2\ell+1}$, where U_{2i+1} is the set of all nodes of ρ_{2i+1} for $i = 0, \ldots, \ell$. By construction, γ satisfies φ, and the lemma is shown. □

By Lemmas 3 and 4, we get the following theorem.

Theorem 3. $\mathcal{M} \models \varphi$ *if and only if there are an* $\mathcal{M}$-sequence γ *of length at most* $|BD(\varphi)|+1$ *and an* $\mathcal{M}^F$ *run* π *such that* π *and* γ *match, and* γ *satisfies* φ*.*

Now we are ready to show membership in NP for the model-checking of $\mathsf{L}(\Diamond)$ formulae on RSMs.

Lemma 5. *$L(\diamond)$ model-checking on* RSM*s is in* NP.

Proof. Let $\mathcal{M}$ be an RSM and φ be a formula. To check in nondeterministic polynomial time if there is a run of $\mathcal{M}^F$ satisfying φ we can guess an $\mathcal{M}$-sequence γ of length at most $|BD(\varphi)| + 1$, then check if it satisfies φ and if there exists a run of $\mathcal{M}^F$ which matches it. The correctness and the time complexity of this algorithm are a consequence of Lemmas 1 and 2, and Theorem 3. □

From Lemma 5 and from the fact that $\mathsf{L}(\diamond)$ model-checking is NP-hard already on finite Kripke structures (see [12]), we have the main result of this section.

Theorem 4. *$L(\diamond)$ model-checking on* RSM*s is* NP*-complete.*

4 Syntactic Fragments of LTL

In this section, we show the complexity bounds on other natural syntactic fragments of LTL. We start introducing a notation for systematically defining such logics.

We denote with $\mathsf{L}(\mathsf{H}_1, \mathsf{H}_2, \ldots)$ the fragment which allows only the temporal operators $\mathsf{H}_1, \mathsf{H}_2, \ldots$. For instance $\mathsf{L}(\mathcal{U})$ is "LTL without $\bigcirc$". Consistently, we have denoted with $\mathsf{L}(\diamond)$ the fragment with only $\diamond$ and $\Box$. For a formula φ, the *temporal height* of φ is the maximum number of nested temporal operators in it. We write $\mathsf{L}^k(\mathsf{H}_1, \ldots)$ to denote the fragment of $\mathsf{L}(\mathsf{H}_1, \ldots)$ where $k \geq 0$ corresponds to the maximum temporal height which is allowed. We omit the index k when no bound is imposed, i.e. $\mathsf{L}(\mathsf{H}_1, \ldots) = \mathsf{L}^\omega(\mathsf{H}_1, \ldots)$.

Formulae of Temporal Height 1

The logic $\mathsf{L}^1(\mathcal{U}, \bigcirc)$ contains all LTL formulae of temporal height 1, that is, all such formulae where temporal operators are not nested.

Model-checking RSMs against formulae of $\mathsf{L}^1(\mathcal{U}, \bigcirc)$ is NP-complete. To show this we observe that since temporal height is 1, formulae of this fragment are boolean combinations of simple formulae with just one temporal operator. Formulae which use only the until operator without nesting it have simple properties as those shown for $\mathsf{L}(\diamond)$ formulae in Section 3. Therefore, we can handle boolean combinations of such formulae as we have done for formulae of $\mathsf{L}(\diamond)$. Being the next operator non nested, we can only use it to specify properties on the second state of a run, and therefore we get membership to NP for $\mathsf{L}^1(\mathcal{U}, \bigcirc)$. Since $L^1(\diamond)$ is already NP-hard for Kripke structures [12], we have the following theorem.

Theorem 5. *$L^1(\mathcal{U}, \bigcirc)$ model-checking of* RSM*s is* NP*-complete.*

As a corollary of the above result and since $L^1(\diamond)$ is NP-hard, we get:

Corollary 1. *Model-checking of* RSM*s for the logics $\mathsf{L}^1(\diamond, \bigcirc)$ and $\mathsf{L}(\mathcal{U})$ is* NP*-complete.*

Fragments with Only "Until" Formulae

In this section, we show that model-checking of RSMs is already EXPTIME-hard for $\mathsf{L}^2(\mathcal{U})$ formulae, and thus for the fragments $\mathsf{L}^{2+k}(\mathcal{U})$, $\mathsf{L}^{2+k}(\mathcal{U},\bigcirc)$ and $\mathsf{L}(\mathcal{U})$. We adapt the proof given in [4] to show hardness for full LTL. We briefly sketch the differences with the original proof and ask the reader to refer to [4] for further details.

Theorem 6. *$\mathsf{L}^2(\mathcal{U})$ model-checking of* RSM*s is* EXPTIME*-complete.*

Proof. Recall that the proof given in [4] consists of a reduction from the membership problem for linearly bounded alternating Turing machines. Given an alternating Turing machine M and an input word w, the reduction constructs a pushdown system A and an $\mathsf{L}(\Diamond,\bigcirc)$ formula φ such that M accepts w if and only if there exists a sequence x accepted by A such that $x \models \varphi$. The idea is that x encodes a computation of M starting with w on the input tape. The formula φ is mainly responsible for checking consistency between consecutive configurations. The automaton A takes care of the tree structure of M computations and ensures that consecutive configurations of the computation tree are encoded consecutively in the word x.

The word x encodes configurations position by position and consecutive configurations are separated by symbol #. Formula φ has the form

$$\Box((p_{\#} \wedge \bigcirc^{n+2} p_{\#}) \Rightarrow (\varphi_1 \wedge \varphi_2 \wedge \varphi_3))$$

where φ_1, φ_2 and φ_3 are boolean combinations involving sub-formulae of the form $\bigcirc^i\psi$ or simple atomic propositions. Note that the condition $(p_{\#} \wedge \bigcirc^{n+2} p_{\#})$ holds true on the beginning of each configuration except the last one. We aim to show that we can replace every sub-formula of the form $\bigcirc^i\psi$ with an until formula, and therefore the reduction will hold for also for the fragment $\mathsf{L}^2(\mathcal{U})$.

For such purpose, let $0, 1, 2$ be fresh atomic propositions. We label all the positions of a configuration with an atomic proposition from $\{0, 1, 2\}$ and require that the first configuration in x is marked with 0, the second with 1, the third with 2, the fourth with 0, and so on. We also require that each position can be distinguished by the other positions in the configuration (i.e., we use pairwise disjoint sets of atomic propositions for encoding the content of each position). We use superscripts on the atomic propositions to denote the position in the configuration.

Consider a φ sub-formula $\bigcirc^j p_a$, $j \leq n$, we recall that according to the encoding from [4], the purpose of this formula is to check that the j-th position of the current configuration contains symbol a. We substitute this formula with $\wedge_{i=0}^{2}(i \rightarrow (i\,\mathcal{U}\, p_a^j))$.

Consider now a φ sub-formula $\bigcirc^{n+j+2} p_a$, $j \leq n$, we recall that the purpose of this formula is to check that the j-th position of the next configuration contains symbol a. We substitute this formula with $\wedge_{i=0}^{2}(i \rightarrow ((i \vee (i+1))\,\mathcal{U}\,(p_a^j \wedge (i+1))))$, where $(i+1)$ is modulo 3.

Note that there are no sub-formulae of φ that relate configurations that are not consecutive, i.e., the index j of a sub-formula $\bigcirc^j p_a$ of φ is not larger than $2n+2$.

Moreover, we do not use more than an until operator for each maximal subformula of nested next operators. Therefore, the formula obtained by translating φ in the above way has temporal height 2. Besides, the size of the new formula is linear in the size of φ. Thus, we have the theorem. □

Thus, from Theorems 2 and 6, we have the following corollary

Corollary 2. *Model-checking of* RSM*s for the logics* $\mathsf{L}^{2+k}(\mathcal{U})$, $\mathsf{L}^{2+k}(\mathcal{U},\bigcirc)$ *and* $\mathsf{L}(\mathcal{U})$*, for each* k*, is* EXPTIME*-complete.*

The Complexity of Nesting "Next" Operator

We consider first the logic $\mathsf{L}^k(\Diamond,\bigcirc)$, that is, the logic containing only formulae over the boolean operators and the temporal operators diamond and next, whose temporal height is bounded by k.

For a formula φ in such fragment we can push the next operators on the atomic propositions with a quadratic blow up. Then, we can replace each maximal next sub-formula with a fresh proposition thus obtaining an $\mathsf{L}^k(\Diamond)$ formula φ' of quadratic size in $|\varphi|$. For a given RSM $\mathcal{M} = (M_1, \ldots, M_h)$ we consider a new RSM $\mathcal{M}'$ such that for each machine M_i we add a machine $M_i^{v_1 \cdots v_k}$ for each sequence $v_1 \ldots v_k$ of $\mathcal{M}$ nodes. Each such machine simulates M_i and maintains in the control the tuple of the next k nodes. Thus nodes, boxes and transitions are added consistently with this meaning. We label each node with the atomic propositions of the corresponding node of $\mathcal{M}$ and with the new atomic propositions corresponding to the next formulae that are satisfied on the sequence of the next k nodes which is paired with it. Observe that since k is constant the size of $\mathcal{M}'$ is polynomial in the size of $\mathcal{M}$. Therefore, model-checking φ on $\mathcal{M}$ reduces in polynomial time to model-checking φ' on $\mathcal{M}'$. Observe that, on formulae of $\mathsf{L}^k(\Diamond)$, for each fixed k, we can determinize the nondeterministic algorithm we have given in the NP membership proof of $\mathsf{L}(\Diamond)$ such that the resulting algorithm runs in polynomial time (recall k is constant). Therefore we have the following theorem.

Theorem 7. $\mathsf{L}^k(\Diamond,\bigcirc)$ *model-checking of* RSM*s is in* PTIME*.*

If we allow nesting the next operator an unbounded number of times, then the model-checking problem becomes NP-complete. Membership in NP is trivial: we just guess a sequence of k nodes and simulate the RSM according to this sequence. NP-hardness is inherited from the complexity of model-checking the same fragment of LTL on finite Kripke structures [6].

Theorem 8. $\mathsf{L}(\bigcirc)$ *model-checking of* RSM*s is* NP*-complete.*

5 Conclusion

In this paper we have analyzed the computational complexity of model-checking fragments of LTL by restricting the choice of the temporal operators and the

Table 1. Model-checking of RSMs against LTL fragments by bounding the temporal height

$0 \leq k < \omega$	**Recursive State Machines**	**Finite Kripke Structures**
$\mathsf{L}(\Diamond)$	NP-complete	NP-complete [12]
$\mathsf{L}^{1+k}(\Diamond)$	NP-complete	NP-complete [12]
$\mathsf{L}(\Diamond,\bigcirc)$	EXPTIME-complete [4]	PSPACE-complete [12]
$\mathsf{L}^{1+k}(\Diamond,\bigcirc)$	NP-complete	NP-complete [6,12]
$\mathsf{L}(\mathcal{U})$	EXPTIME-complete	PSPACE-complete [12]
$\mathsf{L}^{2+k}(\mathcal{U})$	EXPTIME-complete	PSPACE-complete [6]
$\mathsf{L}^{1}(\mathcal{U})$	NP-complete	NP-complete [6]
$\mathsf{L}(\bigcirc)$	NP-complete	NP-complete [6]
$\mathsf{L}^{k}(\bigcirc)$	PTIME	PTIME [6]
$\mathsf{L}(\mathcal{U},\bigcirc)$	EXPTIME-complete [4]	PSPACE-complete [12]
$\mathsf{L}^{2+k}(\mathcal{U},\bigcirc)$	EXPTIME-complete	PSPACE-complete [6]
$\mathsf{L}^{1}(\mathcal{U},\bigcirc)$	NP-complete	NP-complete [6]

temporal height of formulae. The complete picture of the complexity results for the considered LTL fragments is summarized in Table 1. Observe that the model-checking of fragments that are PSPACE-complete on finite Kripke structures becomes EXPTIME-complete on RSMs, instead for all the remaining fragments this problem stays in the same complexity class on either RSMs or finite Kripke structures.

Our main result is showing that $\mathsf{L}(\Diamond)$ is in NP. The argument we use relies on two main properties: the ability of solving in polynomial time a variation of the reachability problem on the RSMs and the possibility of having a small representation for the instances of such problem which we need to check. The first result is not surprising, though it requires some expertise, since the set of reachable configurations in a pushdown system is regular and thus reachability-like queries can be efficiently computed. For the reasons discussed in the introduction, what was not clear was the kind of certificate to use in our arguments and also if a "small" certificate existed for such problem.

In [1], it is shown that LTL model-checking for RSMs and for pushdown systems are inter-reducible in linear time. Moreover, the acceptance problem of linearly bounded alternating Turing machines can be reduced to LTL model-checking on pushdown systems (see [4]). Thus, the EXPTIME-completeness of LTL model-checking problem for RSMs comes directly from pushdown systems. Actually, the proof given in [4] also constitutes a proof of EXPTIME-hardness for $\mathsf{L}(\Diamond,\bigcirc)$.

Investigating logics of formulae with bounded temporal height is interesting and gives a better understanding of the actual complexity of real instances of the model-checking problem. In fact, in the applications, the temporal height often turns out to be at most 2 (or 3 when fairness is involved) even when the specification is quite large and combines a large number of temporal constraints. A bounded height is often invoked as a reason why LTL model-checking is feasible in practice [6]. We think that such investigation for pushdown systems is even

more interesting because of the extremely high complexity of the problem for full LTL.

The logic LTL does not allow to express non regular properties on the runs of an RSM, such as properties of the call-stack. The logic CARET [2] extends LTL with operators that allow for example to check the call-stack content or express properties on the effects of runs within modules (local properties). Model-checking CARET specifications is also EXPTIME-complete. An interesting line of future research is to systematically study the impact of the CARET operators on the complexity of model-checking. It is simple to verify that, except for the operators that refer to the call-stack content, there are no changes with what we have shown in this paper. Remarkably, we can show that model-checking formulae that allow to inspect the stack sequence using the diamond operator is PSPACE-hard, and we conjecture that is also in PSPACE.

References

1. Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T.W., Yannakakis, M.: Analysis of recursive state machines. ACM Trans. Program. Lang. Syst. 27(4), 786–818 (2005)
2. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 467–481. Springer, Heidelberg (2004)
3. Alur, R., Torre, S.L., Madhusudan, P.: Playing games with boxes and diamonds. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 127–141. Springer, Heidelberg (2003)
4. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: Mazurkiewicz, A. W, Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
5. Clarke, E., Kurshan, R.: Computer-aided verification. IEEE Spectrum 33(6), 61–67 (1996)
6. Demri, S., Schnoebelen, P.: The complexity of propositional linear temporal logics in simple cases. Inf. Comput. 174(1), 84–103 (2002)
7. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science. Formal Models and Sematics (B), vol. B, pp. 995–1072 (1990)
8. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for model checking pushdown systems. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 232–247. Springer, Heidelberg (2000)
9. Holzmann, G.J.: The model checker spin. IEEE Trans. Software Eng. 23(5), 279–295 (1997)
10. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems: Specification. Springer, Heidelberg (1991)
11. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57. IEEE Computer Society Press, Los Alamitos (1977)
12. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. J. ACM 32(3), 733–749 (1985)
13. Walukiewicz, I.: Pushdown processes: Games and model-checking. Inf. Comput. 164(2), 234–263 (2001)