Form Methods Syst Des (2010) 36: 65–95
DOI 10.1007/s10703-010-0093-x

Pushdown module checking

Laura Bozzelli · Aniello Murano · Adriano Peron

Published online: 23 March 2010

Abstract *Model checking* is a useful method to verify automatically the correctness of a system with respect to a desired behavior, by checking whether a mathematical model of the system satisfies a formal specification of this behavior. Many systems of interest are open, in the sense that their behavior depends on the interaction with their environment. The model checking problem for finite-state open systems (called *module checking*) has been intensively studied in the literature. In this paper, we focus on *open pushdown systems* and we study the related model-checking problem (*pushdown module checking*, for short) with respect to properties expressed by *CTL* and *CTL** formulas. We show that pushdown module checking against *CTL* (resp., *CTL**) is 2EXPTIME-complete (resp., 3EXPTIME-complete). Moreover, we prove that for a fixed *CTL* or *CTL** formula, the problem is EXPTIME-complete.

Keywords Module checking · Pushdown systems · Branching temporal logics · Tree automata

1 Introduction

In the last decades significant results have been achieved in the area of formal design verification of reactive systems. In particular, a meaningful contribution has been given by algorithmic methods developed in the context of *model-checking* [8, 23, 25]. In this verification method, the behavior of a system, formally described by a mathematical model, is

L. Bozzelli (✉)
IRISA, Campus Universitaire de Beaulieu, Avenue du Général Leclerc, 35042 Rennes Cedex, France
e-mail: laura.bozzelli@dma.unina.it

A. Murano · A. Peron
Dipartimento di Scienze Fisiche, Università di Napoli "Federico II", Via Cintia, 80126 Napoli, Italy

A. Murano
e-mail: murano@na.infn.it

A. Peron
e-mail: peron@na.infn.it

checked against a behavioral constraint specified by a formula in a suitable temporal logic, which enforces either a linear model of time (formulas are interpreted over linear sequences corresponding to single computations of the system) or a branching model of time (formulas are interpreted over infinite trees, which describe all the possible computations of the system). Traditionally, model checking is applied to finite-state systems, typically modeled by labeled state-transition graphs.

In system modeling, we distinguish between *closed* and *open* systems. For a closed system, the behavior is completely determined by the state of the system. For an open system, the behavior is affected both by its internal state and by the ongoing interaction with its environment. Thus, while in a closed system all the nondeterministic choices are internal, and resolved by the system, in an open system there are also external nondeterministic choices, which are resolved by the environment [14]. Model checking algorithms used for the verification of closed systems are not appropriate for the verification of open systems. In the latter case, we should check the system with respect to arbitrary environments and should take into account uncertainty regarding the environment.

Kupferman, Vardi, and Wolper [18] extend model checking from closed finite-state systems to open finite-state systems. In such a framework, the open finite-state system is described by a labeled state-transition graph called *module*, whose set of states is partitioned into a set of *system states* (where the system makes a transition) and a set of *environment states* (where the environment makes a transition). The problem of model checking a module (called *module checking*) has two inputs: a module M and a temporal formula ψ. The idea is that an open system should satisfy a specification ψ no matter how the environment behaves. Let us consider the unwinding of M into an infinite tree, say T_M. Checking whether T_M satisfies ψ is the usual *model-checking problem* [8, 23]. On the other hand, for an open system, T_M describes the interaction of the system with a maximal environment, i.e. an environment that enables all the external nondeterministic choices. In order to take into account all the possible behaviors of the environment, we have to consider all the trees T obtained from T_M by pruning subtrees whose root is a successor of an environment state (pruning these subtrees correspond to disable possible environment choices). Therefore, a module M satisfies ψ if all these trees T satisfy ψ. Note that for the linear-time paradigm, module checking coincides with the usual model checking, since using linear temporal formulas ψ, we require that all the possible interactions of the system with its environment (corresponding to all computations of M, i.e. to all possible full-paths in T_M) have to satisfy ψ. Therefore, while the complexity of model checking for closed and open finite-state systems coincide using linear time logics, when using branching time logics, model checking for open finite-state systems is much harder than model checking for closed finite-state systems. In particular, the problem is EXPTIME-complete for specifications in *CTL* and 2EXPTIME-complete for specifications in *CTL** [18]. Moreover, the complexity of this problem in terms of the size of the module is PTIME-complete.

Recently, the investigation of model-checking techniques has been extended to infinite-state systems. An active field of research is model-checking of closed infinite-state sequential systems. These are systems in which each state carries a finite, but unbounded, amount of information e.g. a pushdown store. The origin of this research is the result of Muller and Schupp that the monadic second-order (*MSO*) theory of graphs induced by pushdown systems is decidable [22]. More recently, Walukiewicz [26] has shown that model checking pushdown systems with respect to *modal μ-calculus* is EXPTIME-complete. Even for a fixed formula in the *alternation-free* modal μ-calculus, the problem is EXPTIME-hard in the size of the pushdown system. The problem remains EXPTIME-complete also for the logic *CTL* [27], which corresponds to a fragment of the alternation-free modal μ-calculus. Recently, in [3], it is showed that even for a fixed *CTL* formula, the problem remains EXPTIME-hard. For the logic *CTL** (which subsumes both *LTL* and *CTL*), the problem is still harder, since it is 2EXPTIME-complete [3]. The situation is quite different for linear-time logics. Model-checking with *LTL* and the *linear-time μ-calculus* is EXPTIME-complete [2]. However, the problem is polynomial-time solvable in the size of the pushdown system.

In the literature, verification of open systems has been also formulated as two-players games. For pushdown systems, games with parity winning conditions are known to be decidable [26]. More recently, in [20], it is shown that pushdown games against *LTL* specifications are 3EXPTIME-complete. This paper contributes to the investigation of model checking of open infinite-state systems by introducing *Open Pushdown systems* (*OPD*, for short) and considering model checking with respect to *CTL* and *CTL**. An *OPD* is a pushdown system in which the set of configurations is partitioned (in accordance with the control state and the symbol on the top of the stack) into a set of *system configurations* and a set of *environment configurations*.

As an example of closed and open pushdown systems, we can consider two drink-dispensing machines (obtained as an extension of the machines defined in [14]). The first machine repeatedly boils water for a while, makes an internal nondeterministic choice and serves either tea or coffee, with the additional constraint that coffee can be served only if the number of coffees served up to that time is smaller than that of teas served. Such a machine can be modeled as a closed pushdown system (the stack is used to guarantee the inequality between served coffees and teas). The second machine repeatedly boils water for a while, asks the environment to make a choice between coffee and tea, and *deterministically* serves a drink according to the external choice, with the additional constraint that coffee can be served only if the number of coffees served up to that time is smaller than that of teas served. Such a machine can be modeled as an open pushdown system. Both machines can be represented by a pushdown system that induces the same infinite tree of possible executions, nevertheless, while the behavior of the first machine depends on internal choices solely, the behavior of the second machine depends also on the interaction with its environment. Thus, for instance, for the first machine, it is always possible to eventually serve coffee. On the contrary, for the second machine, this does not hold. Indeed, if the environment always chooses tea, the second machine will never serve coffee.

We study module checking of (infinite-state) modules induced by *OPD* w.r.t. the branching-time logics *CTL* and *CTL**. First, we note that by results in [22, 28] it easily follows that the considered problems are decidable. Indeed, since pushdown graphs have a decidable *MSO* theory [22] and the unfolding of a graph from a given vertex preserves *MSO* decidability [28], it follows that the tree unfolding of a pushdown system has a decidable *MSO* theory. Now, given a *MSO* specification φ over the tree unfolding $T_{\mathcal{S}}$ of a pushdown system $\mathcal{S}$, it is easy to construct an other *MSO* specification ϕ over $T_{\mathcal{S}}$ such that $T_{\mathcal{S}}$ satisfies ϕ iff the pushdown module checking of $\mathcal{S}$ against φ has a positive answer. Thus, since *CTL* and *CTL** can be effectively translated into *MSO*, the decidability result follows.

In this paper, we establish the exact complexity of pushdown module checking against *CTL* and *CTL** specifications. As in the case of finite-state systems, pushdown module checking is much harder than pushdown model checking for both *CTL* and *CTL**. Indeed, we show that pushdown module checking is 2EXPTIME-complete for *CTL* and 3EXPTIME-complete for *CTL**. We also show that for both *CTL* and *CTL**, the complexity of the problem in terms of the size of the given *OPD* is EXPTIME-complete. For the upper bounds of the complexity results, we exploit the standard automata-theoretic approach. In particular, for *CTL* (resp., *CTL**) we propose an exponential time (resp., a double-exponential time) reduction to the emptiness problem of nondeterministic pushdown tree automata with parity acceptance conditions. The latter problem is known to be decidable in exponential time [19]. Finally, the lower bound for *CTL* (resp., *CTL**) is shown by a technically non-trivial reduction from the word problem for EXPSPACE-bounded (resp., 2EXPSPACE-bounded) alternating Turing Machines.

Outline of the paper. In Sect. 2, we recall the module checking problem as defined in [18] for both *CTL* and *CTL** and define open pushdown systems. In Sect. 3, we recall the framework of *nondeterministic* (*finite-state*) *tree automata* and *nondeterministic pushdown tree automata*, which are exploited in Sect. 4 to solve the pushdown module checking problem against *CTL* and *CTL**. In Sect. 4, we describe algorithms to solve the above mentioned problems, and in Sect. 5, we give lower bounds that match the upper bounds of the proposed algorithms. We conclude in Sect. 6.

2 Preliminaries

2.1 Module checking for branching temporal logics

In this subsection we define the module checking problem for *CTL* and *CTL** [18]. First, we recall syntax and semantics of *CTL* and *CTL**.

Let $\mathbb{N}$ be the set of positive integers. A *tree* T is a prefix closed subset of $\mathbb{N}^*$. The elements of T are called *nodes* and the empty word ε is the *root* of T. For $x \in T$, the set of *children* of x (in T) is $children(T, x) = \{x \cdot i \in T \mid i \in \mathbb{N}\}$. For $k \geq 1$, the (complete) k-ary tree is the tree $\{1, \ldots, k\}^*$. For $x, y \in \mathbb{N}^*$, we write $x \prec y$ to mean that x is a proper prefix of y. For $x \in T$, a (full) path π of T from x is a *minimal* set $\pi \subseteq T$ such that $x \in \pi$ and for each $y \in \pi$ such that $children(T, y) \neq \emptyset$, there is exactly one node in $children(T, y)$ belonging to π. For $y \in \pi$, we denote by π^y the (suffix) path of T from y given by $\{z \in \pi \mid y \preceq z\}$. In the following, for *a path of* T, we mean a path of T from the root ε. For an alphabet Σ, a Σ-labeled tree is a pair $\langle T, V\rangle$, where T is a tree and $V : T \to \Sigma$ maps each node of T to a symbol in Σ.

The logic *CTL** is a branching-time temporal logic [9], where a path quantifier, E ("for some path") or A ("for all paths"), can be followed by an arbitrary linear-time formula, allowing boolean combinations and nesting, over the usual linear temporal operators X ("next"), $\mathcal{U}$ ("until"), F ("eventually"), and G ("always"). There are two types of formulas in *CTL**: *state formulas*, whose satisfaction is related to a specific state (or node of a labeled tree), and *path formulas*, whose satisfaction is related to a specific path. Formally, for a finite set AP of proposition names, the class of state formulas φ and the class of path formulas θ are defined as follows:

$$\varphi := prop \mid \neg\varphi \mid \varphi \wedge \varphi \mid A\,\theta \mid E\,\theta$$

$$\theta := \varphi \mid \neg\theta \mid \theta \wedge \theta \mid X\theta \mid \theta\,\mathcal{U}\,\theta$$

where $prop \in AP$. The set of state formulas φ forms the language *CTL**. The other operators can be introduced as abbreviations in the usual way: for instance, $F\theta$ abbreviates $true\,\mathcal{U}\,\theta$ and $G\theta$ abbreviates $\neg F\neg\theta$. The size $|\varphi|$ of a *CTL** formula φ is the number of distinct subformulas of φ. The *Computation Tree Logic CTL* [8] is a restricted subset of *CTL**, obtained restricting the syntax of path formulas θ as follows: $\theta := X\varphi \mid \varphi\,\mathcal{U}\,\varphi$. This means that X and $\mathcal{U}$ must be immediately preceded by a path quantifier.

We define the semantics of *CTL** (and its fragment *CTL*) with respect to 2^{AP}-labeled trees $\langle T, V\rangle$. Let $x \in T$ and $\pi \subseteq T$ be a path from x. For a state formula φ and a path formula θ, the satisfaction relations $(\langle T, V\rangle, x) \models \varphi$ and $(\langle T, V\rangle, \pi) \models \theta$, meaning that φ holds at node x and θ holds along the path π in $\langle T, V\rangle$, respectively, are defined by induction. The clauses for proposition letters, negation, and conjunction are standard. For the other constructs, we have the following:

- $(\langle T, V\rangle, x) \models A\ \theta$ *iff* for each path π in T from x, $(\langle T, V\rangle, \pi) \models \theta$;
- $(\langle T, V\rangle, x) \models E\ \theta$ *iff* there exists a path π from x such that $(\langle T, V\rangle, \pi) \models \theta$;
- $(\langle T, V\rangle, \pi) \models \varphi$ (where π is a path from x) *iff* $(\langle T, V\rangle, x) \models \varphi$;
- $(\langle T, V\rangle, \pi) \models X\theta$ *iff* $\pi \setminus \{x\} \neq \emptyset$ and $(\langle T, V\rangle, \pi \setminus \{x\}) \models \theta$;[1]
- $(\langle T, V\rangle, \pi) \models \theta_1\, \mathcal{U}\, \theta_2$ *iff* there exists $y \in \pi$ such that $(\langle T, V\rangle, \pi^y) \models \theta_2$ and $(\langle T, V\rangle, \pi^z) \models \theta_1$ for all $z \in \pi$ such that $z \prec y$.

Given a *CTL** (state) formula φ, we say that $\langle T, V\rangle$ satisfies φ if $(\langle T, V\rangle, \varepsilon) \models \varphi$.

In this paper we consider open systems, i.e. systems that interact with their environment and whose behavior depends on this interaction. The (global) behavior of such a system is described by an *open* Kripke structure (called also *module* [18]) $M = \langle AP, W_s, W_e, \rightarrow, w_0, \mu\rangle$, where AP is a finite set of atomic propositions, $W_s \cup W_e$ is a countable set of (global) states partitioned into a set W_s of *system* states and a set W_e of *environment* states (we use W to denote $W_s \cup W_e$), $\rightarrow \subseteq W \times W$ is a (global) transition relation, $w_0 \in W$ is an initial state, and $\mu : W \rightarrow 2^{AP}$ maps each state w to the set of atomic propositions that hold in w. For $w \rightarrow w'$, we say that w' is a successor of w. We assume that the states in M are ordered and the number of successors of each state w, denoted by $bd(w)$, is finite. For each state $w \in W$, we denote by $succ(w)$ the ordered tuple (possibly empty) of w's successors. We say that a state w is *terminal* if it has no successor. When the module M is in a non-terminal system state w_s, then all the states in $succ(w_s)$ are possible next states. On the other hand, when M is in a non-terminal environment state w_e, then the possible next states (that are in $succ(w_e)$) depend on the current environment. Since the behavior of the environment is not predictable, we have to consider all the possible sub-tuples of $succ(w_e)$. The only constraint, since we consider environments that cannot block the system, is that not all the transitions from w_e are disabled.

The set of all (maximal) computations of M starting from the initial state w_0 is described by a W-labeled tree $\langle T_M, V_M\rangle$, called *computation tree*, which is obtained by unwinding M in the usual way. The problem of deciding, for a given branching-time formula ψ over AP, whether $\langle T_M, \mu \circ V_M\rangle$ satisfies ψ, denoted $M \models \psi$, is the usual *model-checking problem* [8, 23]. On the other hand, for an open system, $\langle T_M, V_M\rangle$ corresponds to a very specific environment, i.e. a maximal environment that never restricts the set of its next states. Therefore, when we examine a branching-time specification ψ w.r.t. a module M, ψ should hold not only in $\langle T_M, V_M\rangle$, but in all the trees obtained by pruning from $\langle T_M, V_M\rangle$ subtrees whose root is a child (successor) of a node corresponding to an environment state. The set of these labeled trees is denoted by $exec(M)$, and is formally defined as follows. $\langle T, V\rangle \in exec(M)$ iff $T \subseteq T_M$, V is the restriction of V_M to the tree T, and for all $x \in T$ the following holds:

- if $V_M(x) = w \in W_s$ and $succ(w) = \langle w_1, \ldots, w_n\rangle$, then $children(T, x) = \{x \cdot 1, \ldots, x \cdot n\}$ (note that for $1 \leq i \leq n$, $V(x \cdot i) = V_M(x \cdot i) = w_i$);
- if $V_M(x) = w \in W_e$ and $succ(w) = \langle w_1, \ldots, w_n\rangle$, then there is a sub-tuple $\langle w_{i_1}, \ldots, w_{i_p}\rangle$ of $succ(w)$ such that $children(T, x) = \{x \cdot i_1, \ldots, x \cdot i_p\}$ (note that for $1 \leq j \leq p$, $V(x \cdot i_j) = V_M(x \cdot i_j) = w_{i_j}$), and $p \geq 1$ if $succ(w)$ is not empty.

[1]Note that $\pi \setminus \{x\}$ is a path starting from the unique child of x in π.

Intuitively, each labeled tree $\langle T, V\rangle$ in $exec(M)$ corresponds to a different behavior of the environment. In the following, we consider the trees in $exec(M)$ as 2^{AP}-labeled trees, i.e. taking the label of a node x to be $\mu(V(x))$.

For a module M and a *CTL** (resp., *CTL*) formula ψ, we say that M satisfies ψ, denoted $M \models_r \psi$, if all the trees in $exec(M)$ satisfy ψ. The problem of deciding whether M satisfies ψ is called *module checking* [18]. Note that $M \models_r \psi$ implies $M \models \psi$ (since $\langle T_M, V_M\rangle \in exec(M)$), but the converse in general does not hold. Also, note that $M \not\models_r \psi$ is *not* equivalent to $M \models_r \neg\psi$. Indeed, $M \not\models_r \psi$ just states that there is some tree $\langle T, V\rangle \in exec(M)$ satisfying $\neg\psi$.

2.2 Pushdown module checking

In this paper we consider Modules induced by *Open Pushdown Systems* (*OPD*, for short), i.e., Pushdown systems where the set of configurations is partitioned (in accordance with the control state and the symbol on the top of the stack) into a set of *environment* configurations and a set of *system* configurations.

An *OPD* is a tuple $\mathcal{S} = \langle AP, \Gamma, P, p_0, \Delta, L, Env\rangle$, where AP is a finite set of propositions, Γ is a finite stack alphabet, P is a finite set of (control) states, $p_0 \in P$ is an initial state, $\Delta \subseteq (P \times (\Gamma \cup \{\gamma_0\})) \times (P \times \Gamma^*)$ is a finite set of transition rules (where $\gamma_0 \notin \Gamma$ is the *stack bottom symbol*), $L : P \times (\Gamma \cup \{\gamma_0\}) \to 2^{AP}$ is a labeling function, and $Env \subseteq P \times (\Gamma \cup \{\gamma_0\})$ is used to specify the set of *environment configurations*. A *configuration* is a pair (p, α) where $p \in P$ is a control state and $\alpha \in \Gamma^* \cdot \gamma_0$ is a stack content. We assume that the set $P \times \Gamma^*$ is ordered and for each $(p, A) \in P \times (\Gamma \cup \{\gamma_0\})$, we denote by $next_{\mathcal{S}}(p, A)$ the ordered tuple (possibly empty) of the pairs (q, β) such that $\langle (p, A), (q, \beta)\rangle \in \Delta$. The size $|\mathcal{S}|$ of $\mathcal{S}$ is $|P| + |\Gamma| + |\Delta|$, with $|\Delta| = \sum_{\langle (p,A),(q,\beta)\rangle\in\Delta} |\beta|$. An *OPD* $\mathcal{S}$ induces a module $M_{\mathcal{S}} = \langle AP, W_s, W_e, \to, w_0, \mu\rangle$, where:

- $W_s \cup W_e = P \times \Gamma^* \cdot \gamma_0$ is the set of pushdown configurations;
- W_e is the set of configurations $(p, A \cdot \alpha)$ such that $(p, A) \in Env$;
- $w_0 = (p_0, \gamma_0)$ (initially, the stack is empty);
- $(p, A \cdot \alpha) \to (q, \beta)$ *iff* there is $\langle (p, A), (q, \beta')\rangle \in \Delta$ such that either $A \in \Gamma$ and $\beta = \beta' \cdot \alpha$, or $A = \gamma_0$ (in this case $\alpha = \varepsilon$) and $\beta = \beta' \cdot \gamma_0$ (note that every transition that removes the bottom symbol γ_0 also pushes it back);
- For all $(p, A \cdot \beta) \in W_s \cup W_e$, $\mu(p, A \cdot \beta) = L(p, A)$.

The *pushdown module checking* problem for *CTL* (resp., *CTL**) is to decide, for a given *OPD S* and a *CTL* (resp., *CTL**) formula ψ, whether $\mathcal{M}_{\mathcal{S}} \models_r \psi$.

3 Tree automata

In order to solve the pushdown module checking problem for *CTL* and *CTL**, we use an automata theoretic approach; in particular, we exploit the formalisms of *Nondeterministic* (*finite-state*) *Tree Automata* (*NTA*, for short) [5] and *Nondeterministic Pushdown Tree Automata* (*PD-NTA*, for short) [19].

Nondeterministic (*finite-state*) *tree automata* (NTA). Here we describe *NTA* over (complete) k-ary trees for a given $k \geq 1$. Formally, an *NTA* is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$, where Σ is a finite input alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state,

$\delta : Q \times \Sigma \to 2^{Q^k}$ is a transition function, and F is an acceptance condition. We consider here *Büchi* and *parity* acceptance conditions [5, 11]. In the case of a parity condition, $F = \{F_1, \ldots, F_m\}$ is a finite sequence of subsets of Q, where $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_m = Q$ (m is called the index of $\mathcal{A}$). In the case of a Büchi condition, $F \subseteq Q$.

A run of $\mathcal{A}$ on a Σ-labeled k-ary tree $\langle T, V\rangle$ (where $T = \{1, \ldots, k\}^*$) is a Q-labeled tree $\langle T, r\rangle$ such that $r(\varepsilon) = q_0$ and for each $x \in T$, we have that $\langle r(x \cdot 1), \ldots, r(x \cdot k)\rangle \in \delta(r(x), V(x))$. For a path $\pi \subseteq T$, let $inf_r(\pi) \subseteq Q$ be the set of states that appear as the labels of infinitely many nodes in π. For a parity acceptance condition $F = \{F_1, \ldots, F_m\}$, π is *accepting* if there is an *even* $1 \leq i \leq m$ such that $inf_r(\pi) \cap F_i \neq \emptyset$ and for all $j < i$, $inf_r(\pi) \cap F_j = \emptyset$. For a Büchi condition $F \subseteq Q$, π is *accepting* if $inf_r(\pi) \cap F \neq \emptyset$. A run $\langle T, r\rangle$ is *accepting* if all its paths are accepting. The automaton $\mathcal{A}$ accepts an input tree $\langle T, V\rangle$ iff there is an accepting run of $\mathcal{A}$ over $\langle T, V\rangle$. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of Σ-labeled (complete) k-ary trees accepted by $\mathcal{A}$. The *size* $|\mathcal{A}|$ of an *NTA* $\mathcal{A}$ is $|Q| + |\delta| + |F|$ with $|\delta| = \sum_{(q,\sigma)\in Q\times\Sigma} |\delta(q, \sigma)|$. Note that $|\delta|$ is at most $|\Sigma| \cdot |Q|^{k+1}$.

It is well-known that formulas of *CTL* and *CTL** can be translated into equivalent tree automata (accepting the models of the given formula). In particular, we are interested in optimal translations into parity *NTA*. Fix $k \geq 1$. For the case of *CTL* formulas ψ, first, we can construct according to [17] a Büchi *alternating* (one-way) tree automaton (Büchi *ATA*, for short) $\mathcal{A}_\psi$ over complete k-ary trees with $O(|\psi|)$ states and size $O(k \cdot |\psi|)$ accepting exactly the complete k-ary trees satisfying ψ. Then, by using the construction of Miyano and Hayashi [21], we can convert the Büchi *ATA* $\mathcal{A}_\psi$ into an equivalent Büchi *NTA* with $2^{O(|\psi|)}$ states and size $2^{O(k\cdot|\psi|)}$. Note that Büchi *NTA* correspond to parity *NTA* of index 2. For the case of *CTL** formulas ψ, one can construct [17] an equivalent parity *ATA* $\mathcal{A}_\psi$ over complete k-ary trees with $O(2^{|\psi|})$ states, size $O(k \cdot 2^{|\psi|})$, and index 3.[2] Moreover, a parity (one-way) *ATA* $\mathcal{A}$ over k-ary trees of index m, set of states Q, and transition function δ can be converted, according to [24] (see also [6] for a detailed construction), in single exponential time into an equivalent parity *NTA* $\mathcal{A}_N$ over k-ary trees having index $O(m \cdot |Q|)$, number of states (independent on k) $2^{O(m\cdot|Q|\log(m\cdot|Q|))}$, and size $2^{O(k\cdot m\cdot|Q|\log(m\cdot|Q|))}$. Thus, we obtain the following result.

Lemma 1 [17, 21, 24]

- *Given a* CTL *formula ψ over AP and $k \geq 1$, one can construct a parity* NTA $\mathcal{A}_\psi$ *of size $2^{O(k\cdot|\psi|)}$, index* 2, *and number of states $2^{O(|\psi|)}$ (independent on k) that accepts exactly the set of 2^{AP}-labeled complete k-ary trees that satisfy ψ. Moreover, $\mathcal{A}_\psi$ can be constructed in time $2^{O(k\cdot|\psi|)}$.*
- *Given a* CTL* *formula ψ over AP and $k \geq 1$, we can construct a parity* NTA $\mathcal{A}_\psi$ *of size $2^{O(k\cdot 2^{O(|\psi|)})}$, index $O(2^{|\psi|})$, and number of states $2^{2^{O(|\psi|)}}$ (independent on k) that accepts exactly the set of 2^{AP}-labeled complete k-ary trees that satisfy ψ. Moreover, $\mathcal{A}_\psi$ can be constructed in time $2^{O(k\cdot 2^{O(|\psi|)})}$.*

Nondeterministic pushdown tree automata (PD-NTA). Here, we describe *PD-NTA* (without ε-transitions) over complete k-ary labeled trees. Formally, an *PD-NTA* is a tuple $\mathcal{P} = \langle \Sigma, \Gamma, P, p_0, \rho, F\rangle$, where Σ is a finite input alphabet, Γ is a finite stack alphabet, P is a finite set of (control) states, $p_0 \in P$ is an initial state, $\rho : P \times \Sigma \times (\Gamma \cup \{\gamma_0\}) \to 2^{(P\times\Gamma^*)^k}$ is

[2]Reference [17] gives a translation from *CTL** to *Hesitant alternating tree automata* which are a special case of parity *ATA* of index 3.

a transition function (where $\gamma_0 \notin \Gamma$ is the *stack bottom symbol*), and F is an acceptance condition over P. Intuitively, when the automaton is in state p, reading an input node x labeled by $\sigma \in \Sigma$, and the stack contains a word $A \cdot \alpha$ in $\Gamma^*.\gamma_0$, then the automaton chooses a tuple $\langle (p_1, \beta_1), \dots, (p_k, \beta_k)\rangle \in \rho(p, \sigma, A)$ and splits in k copies such that for each $1 \le i \le k$, a copy in state p_i, and stack content obtained by removing A and pushing β_i, is sent to the node $x \cdot i$ in the input tree.

Formally, a run of the *PD-NTA* $\mathcal{P}$ on a Σ-labeled k-ary tree $\langle T, V\rangle$ (with $T = \{1, \dots, k\}^*$) is a $(P \times \Gamma^*.\gamma_0)$-labeled tree $\langle T, r\rangle$ such that $r(\varepsilon) = (p_0, \gamma_0)$ (initially, the stack is empty) and for each $x \in T$ with $r(x) = (p, A \cdot \alpha)$, there is $\langle (p_1, \beta_1), \dots, (p_k, \beta_k)\rangle \in \rho(p, V(x), A)$ such that for all $1 \le i \le k$, $r(x \cdot i) = (p_i, \beta_i \cdot \alpha)$ if $A \ne \gamma_0$, and $r(x \cdot i) = (p_i, \beta_i \cdot \gamma_0)$ otherwise (note that in this case $\alpha = \varepsilon$).

As with *NTA*, we consider *Büchi* and *parity* acceptance conditions over P. The notion of *accepting* path π is defined as for *NTA* with $\mathit{inf}_r(\pi)$ defined as follows: $\mathit{inf}_r(\pi) \subseteq P$ is the set such that $p \in \mathit{inf}_r(\pi)$ iff there are infinitely many $x \in \pi$ for which $r(x) \in \{p\} \times \Gamma^* \cdot \gamma_0$. A run $\langle T, r\rangle$ is *accepting* if every path $\pi \subseteq T$ is accepting. The *PD-NTA* $\mathcal{P}$ accepts an input tree $\langle T, V\rangle$ iff there is an accepting run of $\mathcal{P}$ over $\langle T, V\rangle$. The language $\mathcal{L}(\mathcal{P})$ of $\mathcal{P}$ contains all and only the trees accepted by $\mathcal{P}$. The *emptiness* problem for *PD-NTA* is to decide, for a given *PD-NTA* $\mathcal{P}$, whether $\mathcal{L}(\mathcal{P}) = \emptyset$.

For a *PD-NTA* $\mathcal{P} = \langle \Sigma, \Gamma, P, p_0, \rho, F\rangle$ with transition function ρ, let ρ_0 be the set of words $\beta \in \Gamma^*.\gamma_0$ occurring in the transition function ρ, i.e., such that there are $(p, \sigma, A) \in P \times \Sigma \times (\Gamma \cup \{\gamma_0\})$ and $\langle (p_1, \beta_1), \dots, (p_k, \beta_k)\rangle \in \rho(p, \sigma, A)$ with $\beta = \beta_i$ for some $1 \le i \le k$. For complexity analysis, we consider the following two parameters: the size $|\rho|$ of ρ given by $|\rho| = \sum_{\langle (p_1,\beta_1),\dots,(p_k,\beta_k)\rangle \in \rho(p,\sigma,A)} |\beta_1| + \dots + |\beta_k|$ and the size $|\rho_0|$ of ρ_0 given by $|\rho_0| = \sum_{\beta \in \rho_0} |\beta|$.

Kupferman et al. in [19] showed that the emptiness problem for a parity *PD-NTA* $\mathcal{P} = \langle \Sigma, \Gamma, P, p_0, \rho, F\rangle$ over k-ary trees can be reduced in polynomial time to the emptiness problem for a parity two-way *ATA* over $|\Gamma|$-ary trees having the same index as $\mathcal{P}$, number of states $O(|P| \cdot |\rho_0|)$ and transition function of size $O(|\rho|)$. Since a parity two-way *ATA* $\mathcal{A}$ over h-ary trees of index m, number of states n, and transition function δ can be converted in time $O(|\delta| \cdot 2^{O(h \cdot n^2 \cdot m \log m)})$ into an equivalent parity *NTA* $\mathcal{A}_N$ over h-ary trees of index $O(m \cdot n)$, number of states $2^{O(n^2 \cdot m \log m)}$, and size $2^{O(h \cdot n^2 \cdot m \log m)}$ [24], and since the emptiness problem for a parity *NTA* $\mathcal{A}_N$ of index m can be solved in time $|\mathcal{A}_N|^{O(m)}$ [10], we obtain the following result.

Proposition 1 [19] *The emptiness problem for a parity* PD-NTA *of index m with n states, stack alphabet Γ, and transition function ρ can be solved in time* $O(|\rho| \cdot 2^{O(|\Gamma| \cdot |\rho_0|^2 \cdot n^2 \cdot m^2 \log m)})$.

A *looping PD-NTA* is a Büchi *PD-NTA* where all the states are accepting. Given a looping *PD-NTA* $\mathcal{P}$ and a parity *NTA* $\mathcal{A}$, one can easily construct in polynomial time a parity *PD-NTA* accepting the intersection of the languages $\mathcal{L}(\mathcal{P})$ and $\mathcal{L}(\mathcal{A})$.

Proposition 2 *For a looping* PD-NTA $\mathcal{P} = \langle \Sigma, \Gamma, P, p_0, \rho, P\rangle$ *and a parity* NTA $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$, *one can construct a parity* PD-NTA $\mathcal{P}'$ *such that* $\mathcal{L}(\mathcal{P}') = \mathcal{L}(\mathcal{P}) \cap \mathcal{L}(\mathcal{A})$. *Moreover, $\mathcal{P}'$ has $|P| \cdot |Q|$ states, the same index as $\mathcal{A}$, the same stack alphabet as $\mathcal{P}$, and its transition function ρ' satisfies*: $|\rho'| = O(|\rho| \cdot |\delta|)$ *and* $\rho'_0 = \rho_0$.

Proof The *PD-NTA* $\mathcal{P}'$ is defined as $\mathcal{P}' = \langle \Sigma, \Gamma, Q \times P, (q_0, p_0), \rho', F'\rangle$ such that $\langle ((q_1, p_1), \beta_1), \dots, ((q_k, p_k), \beta_k)\rangle \in \rho'((q, p), \sigma, A)$ iff $\langle (p_1, \beta_1), \dots, (p_k, \beta_k)\rangle \in \rho(p, \sigma, A)$

and $\langle q_1, \ldots, q_k \rangle \in \delta(q, \sigma)$. Moreover, if $F = \{F_1, \ldots, F_m\}$, then $F' = \{F_1 \times P, \ldots, F_m \times P\}$. $\square$

4 Upper bounds

In this section, we describe algorithms to solve the pushdown module checking against *CTL* and *CTL** which are based on an automata-theoretic approach. As we will see in Sect. 5, the proposed algorithms are asymptotically optimal.

Fix an *OPD* $\mathcal{S} = \langle AP, \Gamma, P, p_0, \Delta, L, Env \rangle$ and a *CTL* (resp., *CTL**) formula ψ. We solve the pushdown module-checking problem for $\mathcal{S}$ against ψ by reducing it to the emptiness of a parity *PD-NTA* $\mathcal{P}_{\mathcal{S}\times\neg\psi}$, which is obtained as the intersection of two tree automata. Essentially, the first automaton, denoted by $\mathcal{P}_{\mathcal{S}}$, is a looping *PD-NTA* that accepts the trees in $exec(M_{\mathcal{S}})$, and the second automaton is a parity *NTA* that accepts the set of trees that do not satisfy ψ. Thus, $M_{\mathcal{S}} \models_r \psi$ iff $\mathcal{L}(\mathcal{P}_{\mathcal{S}\times\neg\psi})$ is empty. The construction proposed here follows (and extends) that given in [18] for solving the module-checking problem for finite-state open systems. The extensions concern the handling of terminal states and the use of pushdown tree automata.

In order to define $\mathcal{P}_{\mathcal{S}}$, we consider an equivalent representation of $exec(M_{\mathcal{S}})$ by complete k-ary trees with $k = \max\{bd(w) \mid w \in W_s \cup W_e\}$ (note that for a pushdown system $\mathcal{S}$, k is finite and can be trivially computed from the transition relation Δ of $\mathcal{S}$). Recall that each tree in $exec(M_{\mathcal{S}})$ is a 2^{AP}-labeled tree that is obtained from $\langle T_{M_{\mathcal{S}}}, V_{M_{\mathcal{S}}} \rangle$ by suitably pruning some of its subtrees. We can encode the tree $\langle T_{M_{\mathcal{S}}}, V_{M_{\mathcal{S}}} \rangle$ as a $2^{AP\cup\{t\}} \cup \{\bot\}$-labeled complete k-ary tree (where $\bot$ and t are fresh proposition names not belonging to AP) in the following way: first, we add the proposition t to the label of all leaf nodes (corresponding to terminal global states) of the tree $T_{M_{\mathcal{S}}}$; second, for each node $x \in T_{M_{\mathcal{S}}}$ with p children $x \cdot 1, \ldots, x \cdot p$ (note that $0 \leq p \leq k$), we add the children $x \cdot (p+1), \ldots, x \cdot k$ and label these new nodes with $\bot$; finally, for each node x labeled by $\bot$ we add recursively k-children labeled by $\bot$. Let $\langle \{1, \ldots, k\}^*, V' \rangle$ be the tree thus obtained. Then, we can encode a tree $\langle T, V \rangle \in exec(M_{\mathcal{S}})$ as the $2^{AP\cup\{t\}} \cup \{\bot\}$-labeled complete k-ary tree obtained from $\langle \{1, \ldots, k\}^*, V' \rangle$ preserving all the labels of nodes of $\langle \{1, \ldots, k\}^*, V' \rangle$ that either are labeled by $\bot$ or belong to T, and replacing all the labels of nodes (together with the labels of the corresponding subtrees) pruned in $\langle T, V \rangle$ with the label $\bot$. In this way, all the trees in $exec(M_{\mathcal{S}})$ have the same structure (they all coincide with $\{1, \ldots, k\}^*$), and they differ only in their labeling. Thus, the proposition $\bot$ is used to denote both "disabled" states and "completion" states, while the proposition t is used to delimitate the prefix of a path, which visits a terminal state, consisting of all and only the nodes which are not labeled by $\bot$. Moreover, since we consider environments that do not block the system, for each node associated with an enabled non-terminal environment state, at least one successor is not labeled by $\bot$. Let us denote by $\widehat{exec}(M_{\mathcal{S}})$ the set of all $2^{AP\cup\{t\}} \cup \{\bot\}$-labeled k-ary trees obtained from $\langle \{1, \ldots, k\}^*, V' \rangle$ in the above described manner. The looping *PD-NTA* $\mathcal{P}_{\mathcal{S}} = \langle \Sigma, \Gamma, P', (p_0, \top), \rho, P' \rangle$, which accepts all and only the trees in $\widehat{exec}(M_{\mathcal{S}})$, is defined as follows:

- $\Sigma = 2^{AP\cup\{t\}} \cup \{\bot\}$;
- $P' = P \times \{\bot, \top, \vdash\}$. From (control) states of the form $(p, \bot)$, $\mathcal{P}_{\mathcal{S}}$ can read only the letter $\bot$, from states of the form $(p, \top)$, it can read only letters in $2^{AP\cup\{t\}}$. Finally, when $\mathcal{P}_{\mathcal{S}}$ is in state $(p, \vdash)$, then it can read both letters in $2^{AP\cup\{t\}}$ and the letter $\bot$. In this last case, it is left to the environment to decide whether the transition to a configuration of the form $((p, \vdash), \alpha)$ is enabled. The three types of (control) states are used to ensure that the environment enables all transitions from enabled system configurations, enables at least

one transition from each enabled non-terminal environment configuration, and disables transitions from disabled configurations.

- The transition function $\rho : P' \times \Sigma \times (\Gamma \cup \{\gamma_0\}) \to 2^{(P'\times\Gamma)^k}$ is defined as follows. According to the definition of P', the automaton $\mathcal{P}_S$ can be in a state of the form $(p, \bot)$, $(p, \top)$, or $(p, \vdash)$. Both in the first and the third cases, $\mathcal{P}_S$ can read $\bot$, which means that the automaton is reading a disabled or a completion node. Thus, independently from the fact that the actual configuration of the automaton is an environment or a system one, ρ propagates states of the form $(p, \bot)$ to all children of the reading node. In case the automaton is in a state of the form $(p, \top)$ or $(p, \vdash)$ and reads a label different from $\bot$, we have to distinguish between different cases. If the reading node is a terminal one, then, all its children must be disabled and so ρ sends to them a k-tuple of states of the type $(p, \bot)$ (independently from the kind of the configuration in which $\mathcal{P}_S$ is). Otherwise, the possible successor states further depend on the particular kind of the configuration in which the automaton is. If $\mathcal{P}_S$ is in a system configuration, then all feasible children of the reading node must not be disabled and so, ρ sends to all of them states of the type $(p, \top)$. If $\mathcal{P}_S$ is in an environment configuration, then all feasible children of the reading node, but one, may be disabled and so, ρ sends to all of them states of the type $(p, \vdash)$, except one, to which ρ sends $(p, \top)$ instead. Formally, let $p \in P$ and $A \in \Gamma \cup \{\gamma_0\}$ with $next_S(p, A) = \langle (p_1, \beta_1), \ldots, (p_d, \beta_d)\rangle$ (with $0 \le d \le k$). Then, for $m \in \{\top, \vdash, \bot\}$ and $\sigma \in \Sigma$, it holds that $\rho((p, m), \sigma, A) \neq \emptyset$ *iff* one of the following holds (where $\alpha = A$ if $A \in \Gamma$, and $\alpha = \varepsilon$ otherwise):
  - $m = \bot$. In this case we have $\sigma = \bot$ and

$$\rho((p, m), \bot, A) = \{\langle \underbrace{((p, \bot), \alpha), \ldots, ((p, \bot), \alpha)}_{k \text{ pairs}} \rangle\}$$

  That is, $\rho((p, m), \bot, A)$ contains exactly one k-tuple. In this case all the successors of the current configuration are disabled.

  - $m = \top$. Then, $\sigma \neq \bot$ and we can have one of the following three cases:

    1. $next_S(p, A)$ is empty (i.e., $d = 0$). In this case $\sigma = L(p, A) \cup \{t\}$ (i.e., the current configuration is terminal) and

$$\rho((p, m), L(p, A) \cup \{t\}, A) = \{\langle ((p, \bot), \alpha), \ldots, ((p, \bot), \alpha)\rangle\}$$

    2. $next_S(p, A)$ is *not* empty (i.e., $d \ge 1$) and $(p, A) \notin Env$. In this case $\sigma = L(p, A)$ and $\rho((p, m), L(p, A), A)$ is given by

$$\{\langle ((p_1, \top), \beta_1), \ldots, ((p_d, \top), \beta_d), \underbrace{((p, \bot), \alpha), \ldots, ((p, \bot), \alpha)}_{k-d \text{ pairs}} \rangle\}$$

    3. $next_S(p, A)$ is *not* empty (i.e., $d \ge 1$) and $(p, A) \in Env$. In this case $\sigma = L(p, A)$ and $\rho((p, m), L(p, A), A)$ is given by

$$\begin{aligned}
&\{\langle ((p_1, \top), \beta_1), ((p_2, \vdash), \beta_1), \ldots, ((p_d, \vdash), \beta_d), ((p, \bot), \alpha), \ldots, ((p, \bot), \alpha)\rangle,\\
&\langle ((p_1, \vdash), \beta_1), ((p_2, \top), \beta_1), \ldots, ((p_d, \vdash), \beta_d), ((p, \bot), \alpha), \ldots, ((p, \bot), \alpha)\rangle,\\
&\vdots\\
&\langle ((p_1, \vdash), \beta_1), ((p_2, \vdash), \beta_1), \ldots, ((p_d, \top), \beta_d), ((p, \bot), \alpha), \ldots, ((p, \bot), \alpha)\rangle\}
\end{aligned}$$

That is, $\rho((p, m), L(p, A), A)$ contains d k-tuples. When the automaton proceeds according to the ith tuple, the environment can disable the transitions to all successors of the current configuration, except the transition associated with the pair (p_i, β_i), which must be enabled.

- $m = \vdash$. Then we can have the following two cases:

1. $\sigma = \perp$. In this case, ρ behaves as in the case $m = \perp$, i.e., we have

$$\rho((p, m), \perp, A) = \{\langle \underbrace{((p, \perp), \alpha), \ldots, ((p, \perp), \alpha)}_{k \text{ pairs}} \rangle\}$$

2. $\sigma \neq \perp$. In this case, ρ behaves as in the case $m = \top$.

Note that $\mathcal{P}_{\mathcal{S}}$ has $3 \cdot |P|$ states, $|\rho|$ is bounded by $k(|P| \cdot |\Gamma| + |\Delta|)$, and $|\rho_0|$ is bounded by $|\Delta|$ (recall that ρ_0 is the set of words $\beta \in \Gamma^*.\gamma_0$ occurring in the transition function ρ and $|\rho_0| = \sum_{\beta \in \rho_0} |\beta|$). Assuming that $|P| \cdot |\Gamma| \leq |\Delta|$, we have that $|\rho| \leq k \cdot |\Delta|$.

We recall that a node labeled by $\perp$ stands for a node that actually does not exist. Thus, we have to take this into account when we interpret *CTL** or *CTL* formulas over trees $\langle T, V \rangle \in \widehat{exec}(M_{\mathcal{S}})$ (where $T = \{1, \ldots, k\}^*$). This means that we have to consider only the paths in $\langle T, V \rangle$ (which we call "legal" paths) that either never visit a node labeled by $\perp$ or contain a *terminal* node (i.e. a node labeled by t). Note that a path is *not* "legal" iff it satisfies the formula $\neg t\, \mathcal{U} \perp$. In order to achieve this, as in [18] we define a function f : *CTL** formulas $\rightarrow$ *CTL** formulas such that $f(\varphi)$ restricts path quantification to only "legal" paths (the function f we consider extends that given in [18], since we have to consider also paths that lead to terminal configurations). The function f is inductively defined as follows:

- $f(prop) = prop$ for any proposition $prop \in AP$;
- $f(\neg\varphi) = \neg f(\varphi)$;
- $f(\varphi_1 \wedge \varphi_2) = f(\varphi_1) \wedge f(\varphi_2)$;
- $f(E\theta) = E((G\neg\perp) \wedge f(\theta)) \vee E((F\ t) \wedge f(\theta))$;
- $f(A\theta) = A((\neg t \mathcal{U} \perp) \vee f(\theta))$;
- $f(X\theta) = X(f(\theta) \wedge \neg\perp)$;
- $f(\theta_1\, \mathcal{U}\, \theta_2) = (f(\theta_1) \wedge \neg\perp)\mathcal{U}(f(\theta_2) \wedge \neg\perp)$.

When φ is a *CTL* formula, the formula $f(\varphi)$ is not necessarily a *CTL* formula, but it has a restricted syntax: its path formulas have either a single linear-time operator or two linear-time operators connected by a Boolean operator. By [15], such formulas have a linear translation to *CTL*.

By definition of f, it follows that for each formula φ and $\langle T, V \rangle \in \widehat{exec}(M_{\mathcal{S}})$, $\langle T, V \rangle$ satisfies $f(\varphi)$ *iff* the 2^{AP}-labeled tree obtained from $\langle T, V \rangle$ removing all the nodes labeled by $\perp$ (and removing the label t) satisfies φ. Therefore, module-checking $\mathcal{S}$ against formula ψ is reduced to check the existence of a tree $\langle T, V \rangle \in \widehat{exec}(M_{\mathcal{S}}) = \mathcal{L}(\mathcal{P}_{\mathcal{S}})$ satisfying $f(\neg\psi)$ (note that $|f(\neg\psi)| = O(|\neg\psi|)$). We reduce the latter to check the emptiness of a parity *PD-NTA* $\mathcal{P}_{\mathcal{S}\times\neg\psi}$ that is defined as the intersection of the looping *PD-NTA* $\mathcal{P}_{\mathcal{S}}$ with a parity *NTA* $\mathcal{A}_{\neg\psi} = \langle \Sigma, Q, q_0, \delta, F \rangle$ accepting exactly the Σ-labeled complete k-ary trees that are models of $f(\neg\psi)$ (recall that $\Sigma = 2^{AP \cup \{t\}} \cup \{\perp\}$). By Lemma 1, if ψ is a *CTL* (resp., *CTL**) formula, then $\mathcal{A}_{\neg\psi}$ has size $2^{O(k \cdot |\psi|)}$ (resp., $2^{O(k \cdot 2^{O(|\psi|)})}$), index 2 (resp., $O(2^{|\psi|})$), and number of states $2^{O(|\psi|)}$ (resp., $2^{2^{O(|\psi|)}}$). Therefore, by Proposition 2, $\mathcal{P}_{\mathcal{S}\times\neg\psi}$ has the same stack alphabet as $\mathcal{S}$ and the following holds:

- For a *CTL* formula ψ, $\mathcal{P}_{\mathcal{S}\times\neg\psi}$ has $O(|P| \cdot 2^{O(|\psi|)})$ states, index 2, and transition function ρ' such that $|\rho'| = O(|\Delta| \cdot 2^{O(k \cdot |\psi|)})$ and $|\rho'_0|$ is bounded by $|\Delta|$.

- For a *CTL** formula ψ, $\mathcal{P}_{\mathcal{S}\times\neg\psi}$ has $O(|P| \cdot 2^{2^{O(|\psi|)}})$ states, index $O(2^{|\psi|})$, and transition function ρ' such that $|\rho'| = |\Delta| \cdot 2^{O(k \cdot 2^{O(|\psi|)})}$ and $|\rho'_0|$ is bounded by $|\Delta|$.

Thus, by Proposition 1 we obtain the following result.

Theorem 1

(1) *The pushdown module-checking problem for* CTL *is in* 2EXPTIME.
(2) *The pushdown module-checking problem for* CTL* *is in* 3EXPTIME.
(3) *For a* fixed CTL *or* CTL* *formula*, *the pushdown module-checking problem is in* EXPTIME.

5 Lower bounds

In this section we give lower bounds for the considered problems that match the upper bounds of the algorithm proposed in Sect. 4. The lower bound for *CTL* (resp., *CTL**) is shown by a reduction from the word problem for EXPSPACE-bounded (resp., 2EXPSPACE-bounded) alternating Turing Machines. Without loss of generality, we consider a model of alternation with a binary branching degree. Formally, an alternating Turing Machine (TM, for short) is a tuple $\mathcal{M} = \langle \Sigma, Q, Q_\forall, Q_\exists, q_0, \delta, F\rangle$, where Σ is the input alphabet, which contains the blank symbol #, Q is the finite set of states, which is partitioned into $Q = Q_\forall \cup Q_\exists$, $Q_\exists$ (resp., $Q_\forall$) is the set of existential (resp., universal) states, q_0 is the initial state, $F \subseteq Q$ is the set of accepting states, and the transition function δ is a mapping $\delta : Q \times \Sigma \to (Q \times \Sigma \times \{L, R\})^2$.

Configurations of $\mathcal{M}$ are words in $\Sigma^* \cdot (Q \times \Sigma) \cdot \Sigma^*$. A configuration $\eta \cdot (q, \sigma) \cdot \eta'$ denotes that the tape content is $\eta\sigma\eta'$, the current state is q, and the reading head is at position $|\eta| + 1$. When $\mathcal{M}$ is in state q and reads an input $\sigma \in \Sigma$ in the current tape cell, then it nondeterministically chooses a triple (q', σ', dir) in $\delta(q, \sigma) = \langle (q_l, \sigma_l, dir_l), (q_r, \sigma_r, dir_r)\rangle$, and then moves to state q', writes σ' in the current tape cell, and its reading head moves one cell to the left or to the right, according to *dir*. For a configuration C, we denote by $succ_l(C)$ and $succ_r(C)$ the successors of C obtained choosing respectively the left and the right triple in $\langle (q_l, \sigma_l, dir_l), (q_r, \sigma_r, dir_r)\rangle$. The configuration C is *accepting* if the associated state q is in F. Given an input $\alpha \in \Sigma^*$, a (finite) computation tree of $\mathcal{M}$ over α is a finite tree in which each node is labeled by a configuration. The root of the tree corresponds to the initial configuration associated with α.[3] An *internal* node that is labeled by a universal configuration C (i.e. the associated state is in $Q_\forall$) has two children, corresponding to $succ_l(C)$ and $succ_r(C)$, while an internal node labeled by an existential configuration C (i.e. the associated state is in $Q_\exists$) has a single child, corresponding to either $succ_l(C)$ or $succ_r(C)$. The tree is accepting iff each its leaf is labeled by an accepting configuration. An input $\alpha \in \Sigma^*$ is *accepted* by $\mathcal{M}$ iff there exists an accepting computation tree of $\mathcal{M}$ over α.

If $\mathcal{M}$ is EXPSPACE-bounded (resp., 2EXPSPACE-bounded), then there is a constant $k \geq 1$ such that for each $\alpha \in \Sigma^*$, the space needed by $\mathcal{M}$ on input α is bounded by $2^{|\alpha|^k}$ (resp., $2^{2^{|\alpha|^k}}$). It is well-known [7] that 2EXPTIME (resp., 3EXPTIME) coincides with the class of all languages accepted by EXPSPACE-bounded (resp., 2EXPSPACE-bounded) alternating Turing Machines.

[3] We assume that initially $\mathcal{M}$'s reading head is scanning the first cell of the tape.

Theorem 2 *Pushdown module checking against* CTL *is* 2EXPTIME-*hard.*

Proof Fix an EXPSPACE-bounded alternating Turing Machine $\mathcal{M} = \langle \Sigma, Q, Q_\forall, Q_\exists, q_0, \delta, F\rangle$ and let $k \geq 1$ be a constant such that for each input $\beta \in \Sigma^*$, the space needed by $\mathcal{M}$ on input β is bounded by $2^{|\beta|^k}$. Moreover, fix an input $\alpha \in \Sigma^*$. We construct an *OPD* $\mathcal{S}$ and a *CTL* formula φ over a finite set *AP* of atomic propositions of sizes *polynomial* in $n = |\alpha|^k$ and in $|\mathcal{M}|$ such that $\mathcal{M}$ accepts α *iff* there is a tree in $exec(M_\mathcal{S})$ that satisfies φ, i.e. *iff* $M_\mathcal{S} \not\models_r \neg\varphi$. Some ideas in the proposed reduction are taken from [16], where there are given lower bounds for the satisfiability of extensions of *CTL* and *CTL**.

Note that any reachable configuration of $\mathcal{M}$ over α can be seen as a word in $\Sigma^* \cdot (Q \times \Sigma) \cdot \Sigma^*$ of length exactly 2^n. If $\alpha = \sigma_1 \ldots \sigma_r$ (where $r = |\alpha|$), then the initial configuration is given by $(q_0, \sigma_1)\sigma_2 \ldots \sigma_r \underbrace{\#\# \ldots \#}_{2^n - r}$.

Let $C = u_1 \ldots u_{2^n}$ be a TM configuration. For each $1 \leq i \leq 2^n$, the value u'_i of the i-th cell of $succ_l(C)$ (resp., $succ_r(C)$) is completely determined by the values u_{i-1}, u_i and u_{i+1} (taking u_{i+1} for $i = 2^n$ and u_{i-1} for $i = 1$ to be some special symbol). We denote by $next_l(u_{i-1}, u_i, u_{i+1})$ (resp., $next_r(u_{i-1}, u_i, u_{i+1})$) our expectation for u'_i (these functions can be trivially obtained from the transition function of $\mathcal{M}$).

First, we give an intuitive explanation of the construction. We use the pushdown module $\mathcal{S}$ to produce all the possible computation trees of $\mathcal{M}$ over the input α. More precisely, each computation tree T of $\mathcal{M}$ over α is associated with a specific environment behavior, i.e., T corresponds to a tree in $exec(M_\mathcal{S})$. External nondeterminism is used in order to produce the actual symbols of each TM configuration. Moreover, existential choices of the TM $\mathcal{M}$ (i.e., the selection between the left and right successor in the current existential TM configuration) are simulated by *external* nondeterminism, and universal choices of $\mathcal{M}$ (i.e., the selection of both left and right successor of the current universal TM configuration) are simulated by *internal* nondeterminism. In both cases, the *OPD* chooses to mark the next guessed TM configuration with a symbol in $\{l, r\}$, where l denotes a left TM successor and r denotes a right TM successor. This ensures that once we fix the environment behavior, we really get a tree where each existential TM configuration is followed by (at least) one TM configuration marked by a symbol in $\{l, r\}$, and every universal configuration is followed (in different branches) by two TM configurations, one marked by the symbol l and the other one marked by the symbol r.

As a sequence of the fact that the TM configurations is produced by external nondeterminism (essentially guessing the next symbol in the configuration), we have to check that the guessed computation tree T (corresponding to environment choices) is a legal computation tree of $\mathcal{M}$ over α. To that purpose, we have to check several properties about each computation path π of T, in particular: (1) each TM configuration along π must have length exactly 2^n, and (2) π is faithful to the evolution of the Turing Machine $\mathcal{M}$, i.e. each TM configuration along π marked by the symbol l (resp., r) must correspond to the left successor $succ_l(C)$ (resp., right successor $succ_r(C)$) of the previous TM configuration C along π. The *OPD* $\mathcal{S}$ cannot guarantee by itself these requirements. Thus, these checks are performed by a suitable *CTL* formula φ. However, in order to construct a *CTL* formula of polynomial size, we need to 'isolate' the (arbitrary) selected path π from the remaining part of the tree. This is the point where we use the pushdown store of the module $\mathcal{S}$. As the TM configurations C are guessed symbol by symbol, they are pushed onto the stack of the *OPD* $\mathcal{S}$. More precisely, each cell of a TM configuration is coded using a block of $n + 1$ symbols of the stack alphabet of $\mathcal{S}$. The first symbol is used to encode the content of the cell and the remaining n

symbols are used to encode the location (the number of cell) on the TM tape (note that the number of cells is in the range $[0, 2^n - 1]$ and can be encoded using n bits).

When the end of an *accepting* computation path π (i.e., a path where the last TM configuration is accepting) is reached, we know that the entire path π is stored on the stack of $\mathcal{S}$. Now, the *OPD* $\mathcal{S}$ can pop the entire computation path π and we can check (by the formula φ) that π is indeed valid. However, in order to check that π is faithful to the evolution of $\mathcal{M}$, we have to check that every location in every configuration is the correct successor of the appropriate location in the previous configuration. To this purpose, we need to individuate for each TM configuration C_1 along π and each TM block bl_1 of C_1, the TM block bl_2 of the previous TM configuration C_2 along π (if any) having the same cell number as bl_1, and check that the equality $u_1 = next_d(u_p, u_2, u_s)$ holds, where: u_1 (resp., u_2) is the cell content of the TM-block bl_1 (resp., bl_2), $d = l$ iff the configuration C_1 is marked by l, and u_s (resp., u_p) is the cell content of the TM block—if any—of C_2 that precedes (resp., follows) bl_2. In order to perform the above described check, the *OPD* $\mathcal{S}$ empties the stack, whose content corresponds to the accepting computation path π, with the additional ability to mark by *internal* nondeterminism exactly one TM block bl_1 with the special symbol $check_1$ ($\mathcal{S}$ keeps track of this symbol by its finite control) and, successively, (in case bl_1 does not belong to the first configuration of π) to mark by *external* nondeterminism exactly one TM block bl_2 by the special symbol $check_2$.

Thus, the *OPD* $\mathcal{S}$ allows to partition the sanity checks of the computation path π into separate branches. In particular, the subtree (called in the following *check-tree*) of the computation tree of $\mathcal{S}$ associated with this phase satisfies the following requirement: the sequence of labels of each branch corresponds to the *reverse* of π with the unique difference that exactly one TM block bl_1 is marked by the symbol $check_1$ and exactly one TM block is marked by the symbol $check_2$. The *OPD S* ensures that bl_1 and bl_2 belongs to two consecutive configurations along π, and bl_1 precedes bl_2 along the branch. Moreover, $\mathcal{S}$ ensures by internal nondeterminism that (independently from the environment choices) all the TM blocks bl_1 of π are checked (i.e., there is a branch whose $check_1$-block corresponds to bl_1). While the *CTL* formula φ *universally* ensures, in particular, that the cell numbers are encoded correctly (hence, each TM configuration of π has length exactly 2^n), and the environment nondeterminism was resolved in the right way, i.e., in each branch selected by the environment the $check_1$-block and the $check_2$-block have the same cell number. Now, we give the details of the construction.

Encoding of configurations and computation paths of $\mathcal{M}$. The set of atomic propositions AP is given by

$$AP = \Sigma \cup (Q \times \Sigma) \cup \{0, 1, \forall, \exists, b, l, r, f, opt, opt_1, opt_2, check_1, check_2\}$$

where 0 and 1 are used to encode the cell number, and the meaning of the letters in $\{\forall, \exists, b, l, r, f, opt, opt_1, opt_2, check_1, check_2\}$ will be explained later. For a TM configuration $C = u_1 u_2 \dots u_k$ (here we do not require that $k = 2^n$), a *pseudo code* of C is a word w over 2^{AP} of the form $\{tag_1\}\{b, u_1\}w_1\{b, u_2\}w_2 \dots \{b, u_k\}w_k\{tag_2\}$, where $w_i \in \{\{0\}, \{1\}\}^n$ for each $1 \le i \le k$, $tag_1 \in \{l, r, f\}$, and $tag_2 = \exists$ if C is an existential configuration, and $tag_2 = \forall$ otherwise. Intuitively, the symbol b is used to mark a TM block, the symbol f is used to mark the initial configuration, while the symbols l and r are used to mark a left and a right TM successor, respectively. If $k = 2^n$, then we say that w is a *code* of C if for each $1 \le i \le 2^n$, w_i corresponds to the binary code of $i - 1$.

Given a finite sequence $\nu = C_1, \dots, C_p$ of TM configurations, a *pseudo code* of ν is a finite word $w_\nu = w_{C_1} \dots w_{C_p}$ such that for each $1 \le i \le p$, w_{C_i} is a pseudo code of C_i, w_{C_1}

is marked by f (i.e., the first symbol of w_{C_1} is $\{f\}$) and each w_{C_i} with $i \neq 1$ is marked by l or r. The word w_ν is a *code* of ν if for each i, w_{C_i} is a code of C_i (note that this implies that C_i has length 2^n). Moreover, we say that w_ν is *faithful the evolution* of $\mathcal{M}$ if for each $1 \leq i < p$, either $w_{C_{i+1}}$ is marked by symbol l and $C_{i+1} = succ_l(C_i)$, or $w_{C_{i+1}}$ is marked by symbol r and $C_{i+1} = succ_r(C_i)$.

Encoding of (accepting finite) computation trees of $\mathcal{M}$ over α. In the following, a *minimal* 2^{AP}-labeled tree is a 2^{AP}-labeled tree such that the children of each node have distinct labels. Moreover, for the ease of presentation, a 2^{AP}-labeled tree $\langle T, V\rangle$ is denoted simply by T. A *pseudo tree-code* is a finite *minimal* 2^{AP}-labeled tree T such that for each path π of T, the associated sequence of labels $w_\pi = w_{C_1} \ldots w_{C_p}$ is the *pseudo code* of some finite sequence $C_1, \ldots, C_p$ of TM configurations, and the following holds:

- C_1 has the form $(q_0, \sigma_1)\sigma_2 \ldots \sigma_r \underbrace{\#\# \ldots \#}_{k}$ (thus, C_1 corresponds to the initial TM configuration associated with α with the exception that the number of blanks # to the right of σ_r can be different from $2^n - r$);
- C_p is accepting and for $1 \leq i < p$, C_i is not accepting;
- for each $1 \leq i < p$ such that C_i is an universal configuration, denoted by x the node of π corresponding to the $\forall$-symbol of w_{C_i}, there is a path π' of T which visits node x and whose associated sequence of labels has the form $w_{\pi'} = w_{C'_1} \ldots w_{C'_i} w_{C'_{i+1}} \ldots$ (note that $w_{C'_j} = w_{C_j}$ for $j \leq i$) such that $w_{C'_{i+1}}$ is marked by l if $w_{C_{i+1}}$ is marked by r, and $w_{C'_{i+1}}$ is marked by r otherwise.

We say that T is a *tree-code* if for each its path π, w_π is a *code* of some sequence of TM configurations. Finally, we say that T is *fair* iff for each its path π, w_π is faithful to the evolution of $\mathcal{M}$. Evidently, each fair tree-code corresponds to some accepting (finite) computation tree of $\mathcal{M}$ over α. Moreover, there is a fair tree-code iff there is an accepting computation tree of $\mathcal{M}$ over α.

Now, we extend a pseudo tree-code T with extra nodes in such a way each leaf x of T is expanded in a tree (called *check tree*), where each branch corresponds to the reverse of the pseudo code (of some accepting sequence of TM configurations) associated with the path of T leading to node x, and marked by additional information. First, we give the definition of check tree.

Check tree: let $w_\nu = w_{C_1} \ldots w_{C_p}$ be a pseudo code of some sequence $\nu = C_1, \ldots, C_p$ of TM configurations. A *check tree* of w_ν is a minimal finite 2^{AP}-labeled tree T_{w_ν} satisfying the following: the root of the tree is labeled by $\{opt\}$ and has two children, one labeled by $\{opt_1\}$ and the other one labeled by $\{opt_2\}$ such that the subtree rooted at the opt_1-child reduces to a unique path whose sequence of labels (excluded the first symbol) is the *reverse* of w_ν, and the subtree T_{opt_2} rooted at the opt_2 child satisfies the following requirements:

- for each path π of T_{opt_2}, the associated sequence of labels (excluded the first symbol) corresponds to the *reverse* of w_ν with the unique difference that there is exactly one TM block bl_1 which is additionally marked by the proposition $check_1$ (i.e., it is of the form $\{b, check_1, u\}w_1$, where $\{b, u\}w_1$ is the corresponding TM block of w_ν) and there is at most one block bl_2 which is additionally marked by the proposition $check_2$. Moreover, the $check_2$-block bl_2 exists iff bl_1 does not belong to the first TM configuration of w_ν, and in this case, bl_1 and bl_2 belong to two consecutive TM configurations, and bl_1 precedes bl_2 along π;
- for each TM block bl of w_ν, there is a path π of T_{opt_2} such the sequence of nodes associated with bl is marked by $check_1$.

The *check-tree* T_{w_ν} is *good* if additionally the following holds:

- for each path π of T_{opt_2} which visits a $check_2$-node, the $check_1$ and $check_2$ TM blocks of π have the same cell number;
- the subtree rooted at a $check_1$-node reduces to a unique path.

Intuitively, the subtree T_{opt_2} of a *good* check-tree of w_ν allows to select for each TM block *bl* non belonging to the first TM configuration of w_ν, the TM block having the same cell number as *bl* and belonging to the previous TM configuration w.r.t. w_ν. Thus, T_{opt_2} is used to check that w_ν is faithful to the evolution of $\mathcal{M}$, while the subtree T_{opt_1} is used to check that w_ν is a in fact a code (i.e. each TM configuration occurring in w_ν has length exactly 2^n and in particular, the cell numbers are encoded correctly).

An *extended pseudo tree-code* is a minimal finite 2^{AP}-labeled tree T_e defined as follows: there is a pseudo tree-code T such that T_e is obtained from T by adding to each leaf x of T a child whose subtree is a check-tree for the sequence of labels associated with the path of T leading to x. If T is a (fair) tree-code, we say that T_e is a (*fair*) *extended tree-code*. Moreover, we say that T_e is *good* if each check-subtree in T_e is good. Thus, there is a good fair extended tree-code iff there is an accepting computation tree of $\mathcal{M}$ over α.

Now, we are ready to construct an *OPD* $\mathcal{S}$ and a *CTL* formula φ such that the set of *finite* trees in $exec(M_\mathcal{S})$ is the set of extended pseudo tree-codes, φ is satisfied only by finite trees, and given an extended pseudo tree-code T, φ is satisfied by T if and only if T is a good fair extended tree-code. Hence, there is an accepting computation tree of $\mathcal{M}$ over α if and only if there is a tree in $exec(M_\mathcal{S})$ which satisfies φ, i.e., if and only if $M_\mathcal{S} \not\models_r \neg\varphi$.

Construction of the OPD $\mathcal{S}$. Here, we describe the main aspects of the behavior of $\mathcal{S}$, ensuring that the set of *finite* trees in $exec(M_\mathcal{S})$ is the set of extended pseudo tree-codes. The formal definition of $\mathcal{S}$ easily follows. In the following, for *state of* $\mathcal{S}$, we mean a pushdown configuration of $\mathcal{S}$ (i.e., a state of the associated module $M_\mathcal{S}$). The *OPD* $\mathcal{S}$ proceeds in three phases.

Phase 1 (*generation of the pseudo-code of the initial TM configuration*): Starting from the initial control state with empty stack content, the *OPD* $\mathcal{S}$ generates by *external* nondeterminism (i.e., the choices are made by the environment) a pseudo code w_C of some TM configuration C pushing it onto the stack (in particular, each transition pushes a symbol of w_C onto the stack) with the additional constraint that C has the form $(q_0, \sigma_1)a_2 \ldots \sigma_r \underbrace{\# \ldots \#}_{k}$ (thus, C corresponds to the initial TM configuration associated with α with the exception that the number of blanks # to the right of σ_r can be different from $2^n - r$). Thus, in this phase, each state reached by $\mathcal{S}$ is an environment state. Moreover, the label of each state (p, β) coincides with the top symbol of the stack content β. Whenever, $\mathcal{S}$ terminates to generate a TM block associated with a blank symbol # to the right of σ_r, $\mathcal{S}$ can choose (by external nondeterminism) to continue to generate an other #-*block*, or to push onto the stack the symbol $tag \in \{\{\exists\}, \{\forall\}\}$ moving to state s, where $tag = \{\forall\}$ and s is a *system* state if $q_0 \in Q_\forall$ (i.e., the guessed first TM configuration is universal), and $tag = \{\exists\}$ and s is an environment state otherwise (i.e., the guessed first TM configuration is existential). From state s, assuming without loss of generality that $q_0 \notin F$, $\mathcal{S}$ moves nondeterministically to an environment state pushing onto the stack a symbol in $\{\{l\}, \{r\}\}$ and switch to phase 2. Thus, in state s, $\mathcal{S}$ simulates the choice of the TM $\mathcal{M}$ from the current guessed first TM configuration.

Phase 2 (*generation of pseudo-codes of TM configurations*): In this phase, $\mathcal{S}$ generates by push transitions pseudo codes of TM configurations as follows: whenever the symbol on the top of the stack is in $\{\{l\}, \{r\}\}$, $\mathcal{S}$ starts to generate repeatedly by *external* nondeterminism (i.e., the states in this phase are environment states) TM blocks bl pushing them onto the stack (in particular, each transition pushes a symbol of bl onto the stack). Moreover, $\mathcal{S}$ keeps track by its finite control if a TM block of the current guessed TM configuration with content in $Q \times \Sigma$ has been already generated. This ensures that at most a block with content in $Q \times \Sigma$ will be generated in this phase. Whenever, $\mathcal{S}$ terminates to generate a TM block and the $Q \times \Sigma$-block bl has been already generated, $\mathcal{S}$ can choose by *external* nondeterminism to terminate the generation of the current guessed TM configuration by pushing on the stack the symbol $tag \in \{\{\exists\}, \{\forall\}\}$ and moving to state $s_q = (p_q, \beta)$, where $tag = \{\forall\}$ and s_q is a *system* state if the Q-state q of bl is in $Q_\forall$ (i.e., the guessed TM configuration is universal), and $tag = \{\exists\}$ and s_q is an environment state otherwise (i.e., the guessed TM configuration is existential). Moreover, depending on whether $q \in F$, $\mathcal{S}$ proceeds as follows:

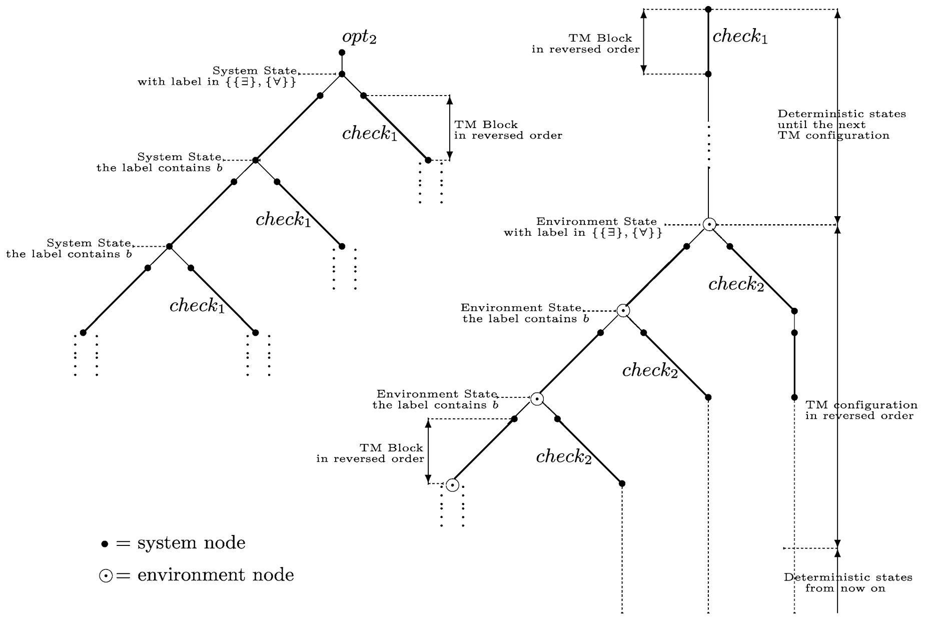
- $q \notin F$: from state s_q, $\mathcal{S}$ moves nondeterministically to an environment state pushing onto the stack a symbol in $\{\{l\}, \{r\}\}$ and repeats phase 2. Thus, in state s_q, $\mathcal{S}$ simulates the choice of the TM $\mathcal{M}$ from the current guessed TM configuration. In particular, system choices (s_q is a system state) correspond to universal choices of $\mathcal{M}$, while environment choices (s_q is an environment state) correspond to existential choices of $\mathcal{M}$.
- $q \in F$: from state $s_q = (p_q, \beta)$, $\mathcal{S}$ without changing the stack content β moves deterministically to the system state (opt, β) (whose label is $\{opt\}$) and switch to phase 3. Note that the reverse of β is the pseudo code of a sequence of TM configurations.

Thus, in phases 1 and 2, a state of $\mathcal{S}$ is a system state iff the top symbol in the associated stack content is $\{\forall\}$ (note that in these states, $\mathcal{S}$ simulates an universal choice of the TM $\mathcal{M}$). Moreover, the label of each state (p, β) with $p \neq opt$ coincides with the top symbol of β. By construction, it follows that if we consider a tree T in $exec(M_\mathcal{S})$ and removes all the subtrees rooted at opt-nodes, then the obtained tree is finite if and only if it is a pseudo tree-code. Moreover, each pseudo tree-code can be obtained in this way.

Phase 3 (*generation of check-trees*): Assume that $\mathcal{S}$ is in the system state (opt, β). By phases 1 and 2, we can also assume that the reverse β^{-1} of the stack content β is the pseudo code of some sequence of TM configurations. Then, from this state $\mathcal{S}$ can choose to move either to the system state (opt_1, β) (whose label is $\{opt_1\}$) or to the system state (opt_2, β) (whose label is $\{opt_2\}$), in both cases without changing the stack content. By selecting opt_1, $\mathcal{S}$ simply empties deterministically the stack by a sequence of pop transitions. The corresponding subtree of the computation tree of $M_\mathcal{S}$ reduces to a finite path whose sequence of labels (excluded the first symbol) is β. By selecting opt_2, $\mathcal{S}$ empties the stack by a sequence of pop transitions with the additional ability to generate by *internal* nondeterminism (i.e., in this phase, each state of $\mathcal{S}$ is a system state) exactly at one TM block bl_1 of β the symbol $check_1$ (more precisely, $\mathcal{S}$ keeps track of this symbol by its finite control) and *successively*, in case bl_1 does not belong to the first TM configuration along β^{-1},[4] to generate by *external* nondeterminism (i.e., after the generation of the $check_1$-symbol, each state of $\mathcal{S}$ is an environment state) exactly at one TM block bl_2 the symbol $check_2$ with the constraint that bl_1 and bl_2 belong to two consecutive TM configurations of β.[5] Figure 1 illustrates the structure of a subtree (of the computation tree of $\mathcal{S}$) rooted at an opt_2-node. Thus, $\mathcal{S}$ ensures that the subtree T of the computation tree of $\mathcal{S}$ rooted at the node associated with (opt, β) satisfies the

[4]Note that $\mathcal{S}$ can check whether this condition is satisfied or not.

[5]In order to ensure that the symbol $check_1$ is generated at least once, we assume that the first TM block of β^{-1}, which is pushed onto the stack in phase 1, is marked by some special symbol.



opt2
System State with label in {{∃}, {∀}}
check1
TM Block in reversed order
System State the label contains b
check1
System State the label contains b
check1
TM Block in reversed order
check1
Deterministic states until the next TM configuration
Environment State with label in {{∃}, {∀}}
check2
Environment State the label contains b
check2
Environment State the label contains b
TM Block in reversed order
check2
TM configuration in reversed order
Deterministic states from now on
• = system node
⊙ = environment node

Fig. 1 Structure of a subtree rooted at an opt_2-node

following: the set of trees obtained from T by disabling some environment choices (without blocking the system) corresponds to the set of check-trees associated with the reverse of β. By construction, it follows that the set of *finite* trees in $exec(M_\mathcal{S})$ coincides exactly with the set of extended pseudo tree-codes.

Construction of the CTL *formula* φ. The main step in the definition of φ is the construction of a *CTL* formula φ_{check} satisfying the following requirement: for each check tree T, T satisfies φ_{check} iff T is good and is associated with a code faithful to the evolution of $\mathcal{M}$. Thus, φ is given by

$$\varphi := (AF\neg EX\ true) \wedge AF(opt \wedge \varphi_{check})$$

where the subformula $(AF\neg EX\ true)$ ensures that each model of φ is a finite tree, and for any extended pseudo tree-code T, the subformula $AF(opt \wedge \varphi_{check})$ requires that T is in fact a good fair extended tree-code. The formula φ_{check} is given by

$$\varphi_{check} := EX(opt_1 \wedge \varphi_{nc}) \wedge EX(opt_2 \wedge \varphi_{good} \wedge \varphi_{fair})$$

Fix a check tree T_{check} of a pseudo code w_ν of some sequence $\nu = C_1, \ldots, C_p$ of TM configurations. Then, φ_{nc} requires that the block numbers in w_ν are encoded correctly (hence, w_ν is a code), φ_{good} requires that T is good, and φ_{fair} requires that w_ν is faithful to the evolution of $\mathcal{M}$. Now, we formally define the *CTL* formulas φ_{nc}, φ_{good}, and φ_{fair}. Let T_{opt_1} and T_{opt_2} be the subtrees rooted at the opt_1-child and opt_2-child of the root of T_{check}, respectively. By definition of φ_{check}, we can assume that φ_{nc} is asserted at the root of T_{opt_1}, and φ_{good} and φ_{fair} are asserted at the root of T_{opt_2}.

First, let us consider the *CTL* formula φ_{nc}. We have to require that w_ν is in fact a code. Recall that T_{opt_1} reduces to a unique path π_ν whose sequence of labels (excluded the first symbol) is the reverse of w_ν. Let us consider two consecutive blocks $bl = cont\{bit_1\}\ldots\{bit_n\}$ and $bl' = cont'\{bit'_1\}\ldots\{bit'_n\}$ along w_ν which belong to the same TM configuration (note that these two blocks appear in reversed order along the unique path π_ν of T_{opt_1}), and let k (resp., k') be the number of cell of the first block bl (resp., the second block bl'), i.e., the integer whose binary code is given by $bit_1\ldots bit_n$ (resp., $bit'_1\ldots bit'_n$).[6] Then, in order to ensure that w_ν is a code, it suffices to require that $k' = (k+1) \bmod 2^n$, $k = 0$ if and only if bl is the first block of the associated TM configuration (i.e., $cont$ is followed along π_ν by a symbol in $\{\{l\},\{r\},\{f\}\}$), and $k' = 2^n$ if bl' is the last block of the associated TM configuration (i.e., bit'_n is preceded along π_ν by a symbol in $\{\{\exists\},\{\forall\}\}$). Therefore, formula φ_{nc} is defined as follows:

$$\begin{aligned}
AG\Bigg\{&\left[((AX)^n b) \rightarrow \left(\left(\bigwedge_{j=0}^{n-1}(AX)^j 0\right) \leftrightarrow (AX)^{n+1}(l \vee r \vee f)\right)\right]\\
&\wedge \left[(\forall \vee \exists) \rightarrow \bigwedge_{j=1}^{n}(AX)^j 1\right]\\
&\wedge \Bigg[\big((AX)^n(b \wedge AX(0 \vee 1))\big)\\
&\qquad \longrightarrow \bigvee_{j=0}^{n-1}\Bigg((AX)^j(1 \wedge (AX)^{n+1}0) \wedge \bigwedge_{i>j}(AX)^i(0 \wedge\ (AX)^{n+1}1)\\
&\qquad\qquad \wedge \bigwedge_{i<j}(AX)^i(1 \leftrightarrow\ (AX)^{n+1}1)\Bigg)\Bigg]\Bigg\}
\end{aligned}$$

Now, we define the formula φ_{good} which ensures that T_{check} is a *good* check-tree of w_ν. Thus, $\varphi_{good} := \varphi_{unique} \wedge \varphi_=$, where φ_{unique} requires that any subtree of T_{opt_2} rooted at a $check_1$-node reduces to a unique path, and $\varphi_=$ requires that for each path π_ν of T_{opt_2} which visits a $check_2$-node, the $check_1$-block bl_1 and the $check_2$-block bl_2 of π_ν have the same cell number. Recall that by definition of check-tree, bl_1 and bl_2 belong to two consecutive TM configurations of w_ν, and bl_1 precedes bl_2 along π_ν. Moreover, a path π_ν of T_{opt_2} visits a $check_2$-node iff the $check_1$-block of π_ν does not belong to the first TM configuration of w_ν (which is marked by f). Also, note that the definition of check-tree (which is in particular a minimal 2^{AP}-labeled tree) ensures that for each node x of T_{opt_2} which has two children and is a descendant of a $check_1$-node, a child of x is marked by $check_2$ and the other one is not marked by $check_2$. Thus, φ_{unique} and $\varphi_=$ are defined as follows:

$$\varphi_{unique} := AG(check_1 \rightarrow AG((AX\ check_2) \vee (AX\neg check_2)))$$

$$\begin{aligned}
\varphi_= := AG\Bigg(&(AX)^n\big(b \wedge check_1 \wedge AF(l \vee r)\big)\\
&\longrightarrow \bigwedge_{j=0}^{n-1}\ \bigvee_{c\in\{0,1\}}\big((AX)^j c \wedge AF(c \wedge (AX)^{n-j}(b \wedge check_2))\big)\Bigg)
\end{aligned}$$

[6]We assume that the first bit is the least significant one.

Note that the correctness in the construction of $\varphi_=$ is ensured by the formula φ_{unique}.

It remains to define the *CTL* formula φ_{fair} which ensures that the sequence of TM configurations $\nu = C_1, \ldots, C_p$ pseudo-encoded by $w_\nu = w_{C_1} \ldots w_{C_p}$ is faithful to the evolution of $\mathcal{M}$ and, in particular, for each $1 \le i < p$, $C_{i+1} = succ_l(C_i)$ if w_{C_i} is marked by l, and $C_{i+1} = succ_r(C_i)$ otherwise. By definition of the *CTL* formulas φ_{nc} and φ_{good}, we can assume that each C_i has length exactly 2^n (and in particular, the cell numbers of the blocks occurring in w_ν are encoded correctly) and T_{check} is a good check-tree associated with w_ν. Thus, in order to ensure that w_ν is faithful to the evolution of $\mathcal{M}$, it suffices to require the following: for each path π_ν of T_{opt_2} containing a $check_2$-node, it holds that $u' = next_d(u_p, u, u_s)$, where u' is the cell content of the $check_1$-block bl_1 of π_ν, u is the cell content of the $check_2$-block bl_2 of π_ν, $d = l$ iff the TM configuration associated with bl_1 is marked by l, and u_s (resp., u_p) is the cell content of the TM block—if any—that precedes (resp., follows) bl_2 along π_ν and belongs to the same TM configuration as bl_2.

Therefore, $\varphi_{fair} := \varphi_{first} \wedge \varphi_{last} \wedge \varphi_{non\text{-}ext}$, where φ_{first} manages the case in which bl_1 is the first block of the associated TM configuration, φ_{last} manages the case in which bl_1 is the last TM block, and $\varphi_{non\text{-}ext}$ manages the remaining cases. For simplicity, we define only $\varphi_{non\text{-}ext}$ (the other two formulas can be defined similarly). Such a formula is defined as follows where $\widetilde{\Sigma} = \Sigma \cup (Q \times \Sigma)$:

$$
\begin{aligned}
AG\Bigg(&\big[b \wedge check_1 \wedge AX(0 \vee 1) \wedge AF(b \wedge (AX)^{n+1} check_2)\big] \\
&\longrightarrow \bigvee_{u_p, u, u_s \in \widetilde{\Sigma}} \bigvee_{d \in \{l, r\}} \big[next_d(u_p, u, u_s) \\
&\qquad\qquad \wedge AF(d \wedge AF(u_s \wedge (AX)^{n+1}(u \wedge check_2 \wedge (AX)^{n+1} u_p)))\big]\Bigg)
\end{aligned}
$$

Correctness in the construction of $\varphi_{non\text{-}ext}$ derives from the fact that since T_{check} is a good check-tree, each subtree rooted at a $check_1$-node reduces to a unique path. This concludes the proof. □

Theorem 3 *Pushdown module checking against* CTL* *is* 3EXPTIME-*hard.*

Proof Let $\mathcal{M} = \langle \Sigma, Q, Q_\forall, Q_\exists, q_0, \delta, F \rangle$ be a 2EXPSPACE-bounded alternating Turing Machine, and let k be a constant such that for each input $\beta \in \Sigma^*$, the space needed by $\mathcal{M}$ on input β is bounded by $2^{2^{|\beta|^k}}$. Fix an input $\alpha \in \Sigma^*$. We build an *OPD* $\mathcal{S}$ and a *CTL** formula φ over a set *AP* of atomic propositions of sizes *polynomial* in $n = |\alpha|^k$ and in $|\mathcal{M}|$ such that $\mathcal{M}$ accepts α *iff* there is a tree in $exec(M_\mathcal{S})$ that satisfies φ, i.e. *iff* $M_\mathcal{S} \not\models_r \neg\varphi$.

Note that any reachable configuration of $\mathcal{M}$ over α can be seen as a word in $\Sigma^* \cdot (Q \times \Sigma) \cdot \Sigma^*$ of length exactly 2^{2^n}. If $\alpha = \sigma_1 \ldots \sigma_r$ (where $r = |\alpha|$), then the initial configuration is given by $(q_0, \sigma_1)\sigma_2 \ldots \sigma_r \underbrace{\#\# \ldots \#}_{2^{2^n} - r}$.

First, we give an intuitive explanation of the construction. As in the proof of Theorem 2, we use the *OPD* $\mathcal{S}$ to produce, for different environment behaviors, all the possible computation trees of $\mathcal{M}$ over the input α. Hence, the computation trees of $\mathcal{M}$ over α correspond to trees in $exec(M_\mathcal{S})$. External nondeterminism is used in order to produce the actual symbols of each TM configuration. Whenever the *OPD* $\mathcal{S}$ reaches the end of an existential (resp., universal) guessed TM configuration C, it simulates the existential (resp., universal) choice of $\mathcal{M}$ from C by external (resp., internal) nondeterminism, and, in particular, $\mathcal{S}$ chooses a symbol in $\{l, r\}$ and marks the next guessed TM configuration with this symbol (where l denotes a left TM successor and r denotes a right TM successor). This ensures that, once we fix the environment behavior, we really get a tree where each existential TM configuration is followed by (at least) one TM configuration marked by a symbol in $\{l, r\}$, and every universal configuration is followed (in different branches) by two TM configurations, one marked by the symbol l and the other one marked by the symbol r. Differently from the proof of Theorem 2, here, we use two counters to encode the number of a TM cell (the encoding is similar to that given in [16]). Since the number of cell is in the range $[0, 2^{2^n} - 1]$, it can be encoded using a 2^n-bit counter. Moreover, we also use an n-bit counter in order to keep track of the *position* (index) of each bit of our 2^n-bit counter. Therefore, each cell of a TM configuration is coded using a block of $2 + (n + 1) \cdot 2^n$, where the first symbol is used to encode the content of the cell, the last symbol is used as separator, and the remaining $(n + 1) \cdot 2^n$ symbols are used to encode the cell number. In particular, this block of $(n + 1) \cdot 2^n$ symbols is a sequence of 2^n sub-blocks of length $n + 1$, where for each $1 \leq i \leq 2^n$, the i-th sub-block is used to encode the value (which is maintained in the first element of the sub-block) and the position (which is given by $i - 1$) of the i-th bit of the 2^n-bit counter.

The (encoding of) TM configurations are guessed, essentially, symbol by symbol and are pushed onto the stack. Note that the *OPD* $\mathcal{S}$ only ensures that each TM sub-block has length $n + 1$. But $\mathcal{S}$ does not guarantee that the TM sub-block numbers and the TM block numbers are encoded correctly. Thus, for the guessed computation tree T (corresponding to the environment choices), we have to check that each computation path π of T is legal, i.e., it satisfies the following requirement: (1) the TM sub-block numbers and the TM blocks numbers are encoded correctly (hence, the number of sub-blocks of each TM block along π is exactly 2^n, and the number of TM blocks of each TM configuration along π is exactly 2^{2^n}), and (2) π is faithful to the evolution of the Turing Machine $\mathcal{M}$, i.e., each TM configuration along π marked by the symbol l (resp., r) must correspond to the left successor $succ_l(C)$ (resp., right successor $succ_r(C)$) of the previous TM configuration C along π.

In order to allow these sanity checks (by a suitable *CTL** formula of polynomial size), whenever the end of an *accepting* computation path π (i.e., a path where the last TM configuration is accepting) is reached, the *OPD* by using both internal and external nondeterminism pop the entire computation path π from the stack. In this way, the *OPD* $\mathcal{S}$ partitions the sanity checks for π into separate branches (corresponding to the reverse of π and augmented with additional information). In particular, when the end of π is pushed onto the stack, $\mathcal{S}$ chooses by internal nondeterminism three options. The first one is used to check that the TM sub-block numbers are encoded correctly, the second one is used to check that the TM block numbers are encoded correctly, and the third one is used to ensure that π is faithful to the evolution of $\mathcal{M}$. Here, we informally describe the behavior of $\mathcal{S}$ when the third option is chosen (the full construction is detailed in the following). Assuming that the TM sub-block and the TM block numbers are encoded correctly, we need to individuate for each TM configuration C_1 along π and each TM block bl_1 of C_1, the TM block bl_2 of the previous TM configuration C_2 along π (if any) having the same cell number as bl_1, and check that the equality $u_1 = next_d(u_p, u_2, u_s)$ holds, where: u_1 (resp., u_2) is the cell content of the TM-block bl_1 (resp., bl_2), $d = l$ iff the configuration C_1 is marked by l, and u_s (resp., u_p) is the cell content of the TM block—if any—of C_2 that precedes (resp., follows) bl_2. Moreover, in order to check that the selected TM blocks bl_1 and bl_2 have the same cell number, we need to partition this check into separate branches, one for each TM sub-block of bl_2.

Thus, the *OPD* $\mathcal{S}$ empties the stack, whose content corresponds to the accepting computation path π, with the additional ability to mark by *internal* nondeterminism exactly one TM block bl_1 with the special symbol $check_1$ ($\mathcal{S}$ keeps track of this symbol by its finite control) and, successively, (in case bl_1 does not belong to the first configuration of π) to mark by *external* nondeterminism the last element of exactly one TM block bl_2 by the special symbol opt_4. Moreover, after having generated the marker opt_4 (note that since bl_2 is popped from the stack, it is processed in reversed order), $\mathcal{S}$ marks by internal nondeterminism exactly one TM sub-block of bl_2 with the special symbol $check_2$. Hence, once we fix the environment choices, the subtree of the computation tree of $\mathcal{S}$ associated with this pop-phase satisfies the following: the sequence of labels of each branch corresponds to the *reverse* of π with the unique difference that exactly one TM block bl_1 is marked by the symbol $check_1$ and exactly one TM sub-block of a TM block bl_2 is marked by the symbol $check_2$. The *OPD S* ensures that bl_1 and bl_2 belongs to two consecutive configurations along π, and bl_1 precedes bl_2 along the branch. Moreover, $\mathcal{S}$ ensures by internal nondeterminism that (independently from the environment choices) all the TM blocks bl_1 of π are checked (i.e., there is a branch whose $check_1$-block corresponds to bl_1), and for each subtree T rooted at a $check_1$-node, if a TM sub-block of a TM block bl_2 is marked by $check_2$, then each sub-block sb_2 of bl_2 is checked (i.e., there is a branch in T whose $check_2$ sub-block corresponds to sb_2). Finally the CTL^* formula φ ensures, in particular, that the environment nondeterminism was resolved in the right way, i.e., in each subtree rooted at a $check_1$-node of a TM block bl_1, there is exactly one opt_4-node, hence, there is exactly one TM block bl_2 whose sub-blocks are marked by $check_2$, and the $check_1$-block bl_1 and the $check_2$-block bl_2 have the same cell number. Now, we give the details of the construction.

Encoding of configurations and (accepting finite) computation trees of $\mathcal{M}$ over α. The set AP of atomic propositions is given by

$$AP = \Sigma \cup (Q \times \Sigma) \cup \{0, 1, \forall, \exists, b, l, r, f, end, \$, opt, check_1, check_2\} \cup \bigcup_{i=1}^{i=4} \{opt_i\}$$

where 0 and 1 are used to encode the cell number, and the meaning of the letters in $\{\forall, \exists, b, l, r, f, end, \$, opt, check_1, check_2\} \cup \bigcup_{i=1}^{i=4}\{opt_i\}$ will be explained later. A TM *sub-block* is a word sb of the form $sb = \{\$, bit\}\{bit_1\}\dots\{bit_n\}$ such that $bit, bit_1, \dots, bit_n \in \{0, 1\}$. The *content* $CON(sb)$ of sb is given by bit (note that the proposition \$ is used to mark the content of sb) and the *sub-block number* $NUM(sb)$ of sb is the integer in $[0, 2^n - 1]$ whose binary code is given by $bit_1 \dots bit_n$. A *pseudo TM block* is a word bl of the form $bl = \{b, u\}sb_1 \dots sb_k\{end\}$ such that $u \in \Sigma \times (Q \times \Sigma)$ and for each $1 \le i \le k$, sb_i is a TM sub-block. The *content* $CON(bl)$ is given by u (note that the proposition b is used to mark the content of bl and end is used as separator). If $k = 2^n$ and for each $1 \le i \le 2^n$, $NUM(sb_i) = i - 1$, we say that bl is a *TM block*. In this case, the *block number* $NUM(bl)$ of bl is the integer in $[0, 2^{2^n}]$ whose binary code is given by $CON(sb_1) \dots CON(sb_{2^n})$.

For a TM configuration $C = u_1 u_2 \dots u_k$ (note that here we do not require that $k = 2^{2^n}$), a *pseudo code* of C is a word w_C of the form $\{tag_1\}bl_1 \dots bl_k\{tag_2\}$ such that for each $1 \le i \le k$, bl_i is a *pseudo TM block* with $CON(bl_i) = u_i$, $tag_1 \in \{l, r, f\}$, and $tag_2 = \exists$ if C is an existential configuration, and $tag_2 = \forall$ otherwise. As in the proof of Theorem 2, the symbol f is used to mark the initial configuration, while the symbols l and r are used to mark a left and a right TM successor, respectively. If $k = 2^{2^n}$ and for each $1 \le i \le k$, bl_i is a *TM block* such that $NUM(bl_i) = i - 1$, then we say that w_C is a *code* of C. The notions of pseudo code and code (faithful to the evolution of $\mathcal{M}$) of a finite sequence of TM configurations are defined as in the proof of Theorem 2, and analogously the notions of pseudo tree-code, tree-code, and fair tree-code. The unique difference is that now we have a different notion of pseudo code or code of a TM configuration. In particular, there is a fair tree-code if and only if there is an accepting computation tree of $\mathcal{M}$ over α.

The notion of check-tree is instead different from that given in the proof of Theorem 2 and is defined as follows. Let $w_\nu = w_{C_1} \dots w_{C_p}$ be a pseudo code of some sequence $\nu = C_1, \dots, C_p$ of TM configurations. Intuitively, as in the proof of Theorem 2, a check tree T of w_ν represents a tree-encoding of w_ν. The goal is to define a CTL^* formula of polynomial size in n and $|\mathcal{M}|$ such that for each check-tree T associated with some pseudo code w_ν, T is satisfied by this formula if and only if w_ν is a code faithful to the evolution of $\mathcal{M}$ and T satisfies some additional properties. Formally, a *check tree* of w_ν is a minimal[7] finite 2^{AP}-labeled tree T_{w_ν} such that the root of the tree is labeled by $\{opt\}$ and has three children, labeled by $\{opt_1\}$, $\{opt_2\}$, and $\{opt_3\}$, respectively. Moreover, the subtree rooted at the opt_1-child reduced to a unique path whose sequence of labels (excluded the first symbol) is the *reverse* of w_ν, and the subtrees T_{opt_2} and T_{opt_3} rooted at the opt_2-child and opt_3-child of the root, respectively, satisfy the following requirements:

- Properties of T_{opt_2}: for each path π of T_{opt_2}, the associated sequence of labels (excluded the first symbol) corresponds to the *reverse* of w_ν with the unique difference that there is exactly one pseudo TM block bl_1 of w_ν which is additionally marked by the proposition $check_1$ (in particular, each symbol in the pseudo block is marked by $check_1$) and there is at most one *TM sub-block* sb_2 which is additionally marked by the proposition $check_2$. The $check_2$-sub-block sb_2 exists iff bl_1 is not the first pseudo TM block of w_ν, and in this case, bl_1 and the pseudo block of sb_2 are consecutive, and bl_1 precedes sb_2 along π. Moreover, the following holds:
  - for each pseudo TM block bl of w_ν, there is a path π of T_{opt_2} such that the sequence of nodes associated with bl is marked by $check_1$;
  - let T_1 be a subtree rooted at a $check_1$-node associated with a pseudo block bl_1. If bl_1 is preceded by the pseudo block bl_2 along w_ν, then for each sub-block sb_2 of bl_2, there is a path in T_1 such that the sequence of nodes associated with sb_2 is marked by $check_2$.

  Note that T_{opt_2} is uniquely determined unless tree isomorphisms.
- Properties of T_{opt_3}: for each path π of T_{opt_3}, the associated sequence of labels (excluded the first symbol) corresponds to the *reverse* of w_ν with the unique difference that there is exactly one pseudo TM block bl_1 of w_ν which is additionally marked by the proposition $check_1$ (in particular, each symbol in the pseudo block is marked by $check_1$), there is at most one TM pseudo-block bl_2 such that the last symbol $\{end\}$ of bl_2 is additionally marked by proposition opt_4, and there is exactly one sub-block of bl_2 which is additionally marked by proposition $check_2$. Moreover, the opt_4-pseudo-block bl_2 exists iff bl_1 does not belong to the first TM configuration of w_ν, and in this case, bl_1 and bl_2 belong to two consecutive TM configurations, and bl_1 precedes bl_2 along π. Moreover, the following holds:
  - for each pseudo TM block bl of w_ν, there is a path π of T_{opt_3} such the sequence of nodes associated with bl is marked by $check_1$;
  - let T_1 be a subtree rooted at a opt_4-node, associated with the last symbol of a pseudo block bl_2 (recall that bl_2 appears in reversed order along a path of T_{opt_3}). Then, for each sub-block sb_2 of bl_2, there is a path in T_1 such that the sequence of nodes associated with sb_2 is marked by $check_2$.

The *check-tree* T_{w_ν} is *good* if additionally each pseudo TM block in w_ν is in fact a TM block and T_{opt_3} satisfies the following:

[7]Recall that a minimal 2^{AP}-labeled tree is a 2^{AP}-labeled tree such that the children of each node have distinct labels.

- for each $check_1$-node x of T_{opt_3} (associated with a symbol of a TM block bl), the subtree rooted at x contain at most one opt_4-node. Moreover, if such opt_4-node exists (note that this holds iff bl does not belong to the first TM configuration of w_ν), then the TM block associated with the opt_4-node has the same block-number as bl.

Intuitively, the subtree T_{opt_1} of a check-tree T_{w_ν} is used to ensure that each pseudo TM block in w_ν is a in fact a TM block, while the subtree T_{opt_2} is used to ensure that w_ν is a code (i.e. each TM configuration occurring in w_ν has length exactly 2^{2^n} and in particular, the block numbers are encoded correctly). Finally, the subtree T_{opt_3} of a *good* check-tree of w_ν allows to select for each TM block bl non belonging to the first TM configuration of w_ν, the TM block having the same block number as bl and belonging to the previous TM configuration w.r.t. w_ν. Thus, T_{opt_3} is used to ensure that w_ν is faithful to the evolution of $\mathcal{M}$.

An *extended pseudo tree-code* is a minimal finite 2^{AP}-labeled tree T_e defined as follows: there is a pseudo tree-code T such that T_e is obtained from T by adding to each leaf x of T a child whose subtree is a check-tree for the sequence of labels associated with the path of T leading to x (recall that this sequence of labels is the pseudo code of some sequence of TM configurations). If T is a (fair) tree-code, we say that T_e is a (*fair*) *extended tree-code*. Moreover, we say that T_e is *good* if each check-subtree in T_e is good. Thus, there is a good fair extended tree-code iff there is an accepting computation tree of $\mathcal{M}$ over α.

Now, we are ready to construct an *OPD* $\mathcal{S}$ and a *CTL** formula φ such that the set of *finite* trees in $exec(M_\mathcal{S})$ is the set of extended pseudo tree-codes, φ is satisfied only by finite trees, and given an extended pseudo tree-code T, φ is satisfied by T if and only if T is a good fair extended tree-code. Hence, there is an accepting computation tree of $\mathcal{M}$ over α if and only if there is a tree in $exec(M_\mathcal{S})$ which satisfies φ, i.e., if and only if $M_\mathcal{S} \not\models_r \neg\varphi$.

Construction of the OPD $\mathcal{S}$. Here, we describe the main aspects of the behavior of $\mathcal{S}$, ensuring that the set of *finite* trees in $exec(M_\mathcal{S})$ is the set of extended pseudo tree-codes. The formal definition of $\mathcal{S}$ easily follows. In the following, for *state of* $\mathcal{S}$, we mean a pushdown configuration of $\mathcal{S}$ (i.e., a state of the associated module $M_\mathcal{S}$). The *OPD* $\mathcal{S}$ proceeds in three phases.

Phase 1 (*generation of the pseudo-code of a sequence of TM configuration*): The behavior of $\mathcal{S}$ in this phase is very similar to the behavior of the *OPD* (in phases 1 and 2) in the proof of Theorem 2. In particular, the *OPD* $\mathcal{S}$ guesses a pseudo code of some sequence of TM configurations pushing it onto the stack with the additional constraint that the first guessed TM configuration C has the form $(q_0, \sigma_1)\sigma_2 \ldots \sigma_r \underbrace{\# \ldots \#}_{k}$ (thus, C corresponds to the initial TM configuration associated with α with the exception that the number of blanks # to the right of σ_r can be different from $2^{2^n} - r$). As the *OPD* in the proof of Theorem 2, in this phase, a state of $\mathcal{S}$ is a system state iff the top symbol in the associated stack content is $\{\forall\}$ (in these states, $\mathcal{S}$ simulates the choice of the TM $\mathcal{M}$ from the current guessed universal TM configuration). Thus, universal choices of the TM $\mathcal{M}$ are simulated by system choices of $\mathcal{S}$ from (system) states with stack top symbol $\forall$, while existential choices of the TM $\mathcal{M}$ are simulated by environment choices of $\mathcal{S}$ from (environment) states with stack top symbol $\exists$. Whenever, $\mathcal{S}$ terminates to generate on the stack the last symbol ($\exists$ or $\forall$) of the pseudo code of an accepting TM configuration, then $\mathcal{S}$ moves deterministically (without changing the stack content) to a system state of the form (opt, β) whose label is $\{opt\}$ and switch to phase 2. Note that in this phase, the label of each state (p, β) with $p \neq opt$ coincides with the top symbol of β. Thus, as for the *OPD* in the proof of Theorem 2, if we consider a tree T in $exec(M_\mathcal{S})$ and removes all the subtrees rooted at *opt*-nodes, then the obtained tree is finite if and only if it is a pseudo tree-code. Moreover, each pseudo tree-code can be obtained in this way.

Phase 2 (*generation of check-trees*): Assume that $\mathcal{S}$ is in the system state (opt, β). By phase 1, we can also assume that the reverse β^{-1} of the stack content β is the pseudo code of some sequence of TM configurations. Then, from this state $\mathcal{S}$ can choose to move or to the system state (opt_1, β) (whose label is $\{opt_1\}$), or to the system state (opt_2, β) (whose label is $\{opt_2\}$), or to the system state (opt_2, β) (whose label is $\{opt_2\}$), in both cases without changing the stack content.

By selecting opt_1, $\mathcal{S}$ simply empties deterministically the stack by a sequence of pop transitions. The corresponding subtree of the computation tree of $M_\mathcal{S}$ reduces to a finite path whose sequence of labels (excluded the first symbol) is β.

By selecting opt_2, $\mathcal{S}$ empties the stack by a sequence of pop transitions with the additional ability to generate by *internal* nondeterminism (i.e., in this phase, each state of $\mathcal{S}$ is a system state) exactly at one pseudo TM block bl_1 of β the symbol $check_1$ (more precisely, $\mathcal{S}$ keeps track of this symbol by its finite control) and *successively*, in case bl_1 is not the first TM pseudo block of β^{-1},[8] to generate by *internal* nondeterminism exactly at one TM sub-block sb_2 the symbol $check_2$ with the constraint that bl_1 and the pseudo block of sb_2 are consecutive.[9] Thus, after having selected opt_2, each state reached by $\mathcal{S}$ is a system state.

Finally, by selecting opt_3, $\mathcal{S}$ empties the stack by a sequence of pop transitions with the additional ability to generate by *internal* nondeterminism (i.e., in this phase, each state of $\mathcal{S}$ is a system state) exactly at one pseudo TM block bl_1 of β the symbol $check_1$ and *successively*, in case bl_1 does not belong to the first TM configuration along β^{-1},[10] to generate by *external* nondeterminism at the last element of exactly one TM block bl_2 the symbol opt_4 with the constraint that bl_1 and bl_2 belong to two consecutive TM configurations of β. Moreover, after having generated, the marker opt_4 at the last symbol of bl_2 (note that bl_2 appears in reverse order along β), $\mathcal{S}$ generates by *internal* nondeterminism exactly at one sub-block of bl_2 the marker $check_2$. Figures 2 and 3 illustrate the structure of a subtree (of the computation tree) rooted at an opt_3-node.

Thus, $\mathcal{S}$ ensures that the subtree T of the computation tree of $\mathcal{S}$ rooted at the node associated with (opt, β) satisfies the following: the set of trees obtained from T by disabling some environment choices (without blocking the system) corresponds to the set of check-trees associated with the reverse of β. By construction it follows that the set of *finite* trees in $exec(M_\mathcal{S})$ coincides exactly with the set of extended pseudo tree-codes.

Construction of the CTL* *formula* φ. The main step in the definition of φ is the construction of a *CTL** formula φ_{check} satisfying the following requirement: for each check tree T, T satisfies φ_{check} iff T is good and is associated with a code faithful to the evolution of $\mathcal{M}$. Thus, φ is given by

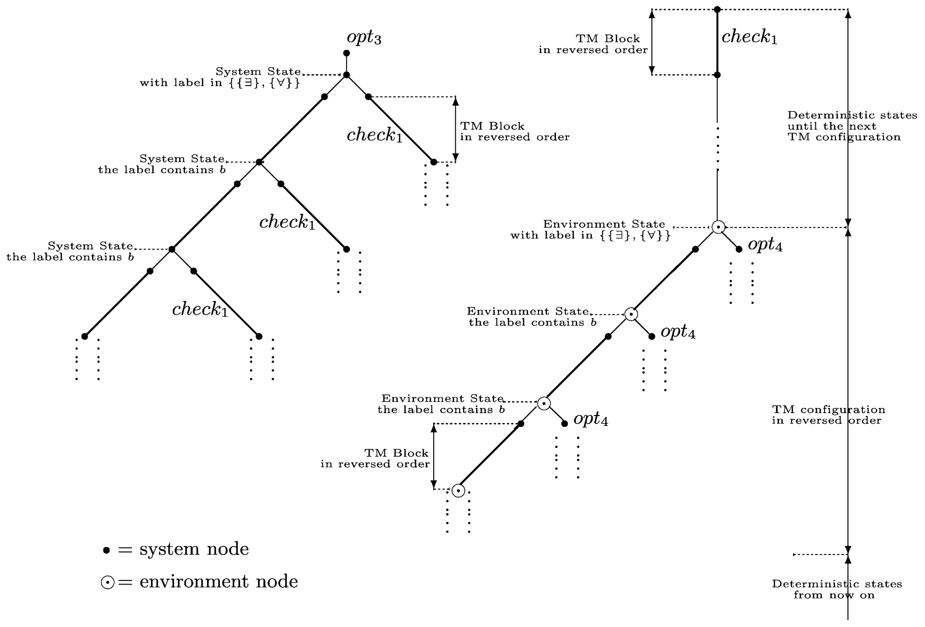
$$\varphi := (AF\neg EX\ true) \wedge AF(opt \wedge \varphi_{check})$$

where the subformula $(AF\neg EX\ true)$ ensures that each model of φ is a finite tree, and for any extended pseudo tree-code T, the subformula $AF(opt \wedge \varphi_{check})$ requires that T is in fact

[8]Note that $\mathcal{S}$ can check whether this condition is satisfied or not.

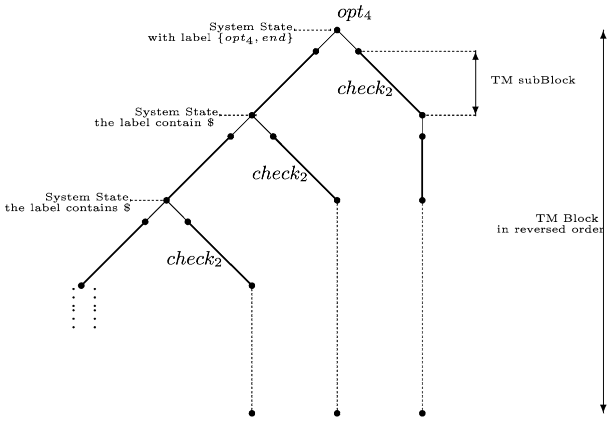
[9]In order to ensure that the symbol $check_1$ is generated at least once, we assume that the first TM block of β^{-1}, which is pushed onto the stack in phase 1, is marked by some special symbol.

[10]Note that $\mathcal{S}$ can check whether this condition is satisfied or not.



opt_3
System State with label in $\{\{\exists\},\{\forall\}\}$
System State the label contains b
System State the label contains b
$check_1$
$check_1$
$check_1$
TM Block in reversed order
TM Block in reversed order
$check_1$
Deterministic states until the next TM configuration
Environment State with label in $\{\{\exists\},\{\forall\}\}$
opt_4
Environment State the label contains b
opt_4
Environment State the label contains b
opt_4
TM Block in reversed order
TM configuration in reversed order
Deterministic states from now on
• = system node
⊙ = environment node

Fig. 2 Structure of a subtree rooted at an opt_3-node



opt_4
System State with label $\{opt_4, end\}$
System State the label contain \$
System State the label contain \$
$check_2$
$check_2$
$check_2$
TM subBlock
TM Block in reversed order

Fig. 3 Structure of a subtree rooted at an opt_4-node.

a good fair extended tree-code. The formula φ_{check} is given by

$$\varphi_{check} := EX(opt_1 \wedge \varphi_{block}) \wedge EX(opt_2 \wedge \varphi_{code}) \wedge EX(opt_3 \wedge \varphi_{good} \wedge \varphi_{fair})$$

Fix a check tree T_{check} of a pseudo code w_ν of some sequence $\nu = C_1, \ldots, C_p$ of TM configurations. Then, φ_{block} requires that each pseudo TM block in w_ν is in fact a TM block, φ_{code} requires that the block numbers in w_ν are encoded correctly (hence, w_ν is a code), φ_{good} requires that T is good, and φ_{fair} requires that w_ν is faithful to the evolution of $\mathcal{M}$. Since the construction of the CTL^* formula φ_{block} is very similar to the construction of formula φ_{nc} in the proof of Theorem 2, here, we focus on formulas φ_{code}, φ_{good}, and φ_{fair}. Let T_{opt_2} and T_{opt_3} be the subtrees rooted at the opt_2-child and opt_3-child of the roof of T_{check}, respectively. By definition of φ_{check}, we can assume that φ_{code} is asserted at the root of T_{opt_2}, and φ_{good} and φ_{fair} are asserted at the root of T_{opt_3}.

First, let us consider the CTL^* formula φ_{code}. By using φ_{block}, we can assume that each pseudo block in w_ν is in fact a block. Fix two consecutive blocks bl_1 and bl_2 along w_ν such that bl_1 and bl_2 belong to the same TM configuration and bl_2 precedes bl_1 along w_ν. We have to require that $NUM(bl_2) < 2^{2^n} - 1$ and $NUM(bl_1) = NUM(bl_2) + 1$, $NUM(bl_2) = 0$ if bl_2 is the first TM block of the associated TM configuration (i.e., bl_2 is preceded along w_ν by a symbol in $\{\{l\}, \{r\}, \{f\}\}$), and $NUM(bl_1) = 2^{2^n}$ if bl_1 is the last TM block of the associated TM configuration (i.e., bl_1 is followed along w_ν by a symbol in $\{\{\exists\}, \{\forall\}\}$). Note that the requirement $NUM(bl_2) < 2^{2^n} - 1$ and $NUM(bl_1) = NUM(bl_2) + 1$ is equivalent to the following:

- there is a sub-block sb_1 of bl_1 such that denoted by sb_2 the sub-block of bl_2 having the same sub-block number as sb_2, it holds that $CON(sb_1) = 1$ and $CON(sb_2) = 0$. Moreover, for each sub-block $sb'_1 \neq sb_1$ of bl_1, denoted by sb'_2 the sub-block of bl_2 having the same sub-block number as sb'_1, the following holds: $CON(sb'_1) = 0$ and $CON(sb'_2) = 1$ if sb'_1 precedes sb_1 along w_ν, and $CON(sb_1) = CON(sb'_1)$ otherwise.

By definition of check tree, there is a path π_ν of T_{opt_2} such that the block bl_1 is marked by $check_1$ and the subtree T' rooted at the node of π_ν associated with the *end*-symbol of bl_1 (recall that bl_1 precedes bl_2 along π_ν and appears in reversed order) satisfies the following: (1) for each path of T', there is exactly one sub-block marked by $check_2$; moreover, this sub-block belongs to bl_2, and (2) for each sub-block sb_2 of bl_2, there is a path of T' such that the sequence of nodes associated with sb_2 is marked by $check_2$. Moreover, note that each $check_1$-node has exactly one child. Thus, formula φ_{code} is defined as follows:

$$\varphi_{code} := \varphi_{first} \wedge \varphi_{last} \wedge \varphi_{inc}$$

where

$$\varphi_{first} := AG\Big(\big[end \wedge (\neg b\,\mathcal{U}\,(b \wedge X(l \vee r \vee f)))\big] \rightarrow \big[((\$ \rightarrow 0) \wedge \neg b)\,\mathcal{U}\,b\big]\Big)$$

$$\varphi_{last} := AG\Big(\big[\forall \vee \exists\big] \rightarrow \big[((\$ \rightarrow 1) \wedge \neg b)\,\mathcal{U}\,b\big]\Big)$$

$$\begin{aligned}\varphi_{inc} := AG\Bigg(&\big[end \wedge check_1 \wedge (\neg b\,\mathcal{U}\,(b \wedge \neg X(l \vee r \vee f)))\big]\\ &\longrightarrow \Bigg[\Bigg\{(X^n\$) \rightarrow \bigvee_{bit\in\{0,1\}} \xi(bit, bit)\Bigg\}\,\mathcal{U}\,\Big\{(X^n\$) \wedge \xi(1,0)\\ &\qquad \wedge X^{n+1}\big(((X^n\$) \rightarrow \xi(0,1))\,\mathcal{U}\,(b \wedge check_1)\big)\Big\}\Bigg]\Bigg)\end{aligned}$$

where $\xi(bit_1, bit_2)$ is defined as follows:

$$\begin{aligned}\xi(bit_1, bit_2) := E\Bigg(&(X^n bit_1) \wedge F(check_2 \wedge \$ \wedge bit_2)\\ &\wedge \bigwedge_{j=0}^{n-1} \bigvee_{bit\in\{0,1\}} ((X^j bit) \wedge F(bit \wedge check_2 \wedge X^{n-j}\$))\Bigg)\end{aligned}$$

Now, we define the formula φ_{good} which ensures that T_{check} is a *good* check-tree of w_ν. Thus, $\varphi_{good} := \varphi_{unique} \wedge \varphi_{=}$, where φ_{unique} requires that each subtree of T_{opt_3} rooted at a $check_1$-node associated with some TM block bl of w_ν contains at most one opt_4-node, and $\varphi_{=}$ additionally requires that in case the opt_4-node exists (i.e., bl does not belong to the first TM configuration along w_ν), then the TM block associated with the opt_4-node has the same block number as bl. Note that the definition of check-tree (which is in particular a minimal 2^{AP}-labeled tree) ensures that the first requirement is equivalent to the following: for each subtree of T_{opt_3}, there is no node having a child labeled by opt_4 and an other child which is not labeled by opt_4. Thus, φ_{unique} is defined as follows (recall that we can assume that φ_{good} is asserted at the root of T_{opt_3}):

$$\varphi_{unique} := AG\neg((EX\ opt_4) \wedge (EX\neg opt_4))$$

Now, let us consider $\varphi_{=}$. Let T_1 be a subtree rooted at a $check_1$-node and whose label contains *end* (i.e., the node corresponds to the last symbol of a TM block bl) and assume that bl does not belong to the first TM configuration. The, by using formula φ_{unique}, we can assume that T_1 contains exactly one opt_4-node (by def. of check tree, the label of this node contain *end*). Let bl' be the TM block associated with the opt_4-node. Then, we have to require that bl and bl' have the same block-number, i.e., for each sub-block sb of bl, it holds that $CON(sb') = CON(sb)$, where sb' is the sub-block of bl' having the same sub-block number as sb. Now, the definition of check tree ensures that the subtree T_{opt_4} rooted at the considered opt_4-node satisfies the following: (1) for each path of T_{opt_4}, there is exactly one sub-block marked by $check_2$; moreover, this sub-block belongs to bl', and (2) for each sub-block sb' of bl', there is a path of T_{opt_4} such that the sequence of nodes associated with sb' is marked by $check_2$. Moreover, each $check_1$-node has exactly one child. Thus, formula $\varphi_{=}$ is defined as follows:

$$\varphi_{=} := AG\left((end \wedge check_1 \wedge F(l \vee r)) \longrightarrow G\left((check_1 \wedge (X^n\$)) \rightarrow \bigvee_{bit \in \{0,1\}} \xi(bit, bit)\right)\right)$$

where $\xi(bit_1, bit_2)$ is the subformula of φ_{inc} defined above. Note that the correctness of the construction is crucially based on the fact that each subtree rooted at a $check_1$-node contains at most one opt_4-child.

It remains to define the CTL^* formula φ_{fair} which ensures that the sequence of TM configurations $\nu = C_1, \ldots, C_p$ pseudo-encoded by $w_\nu = w_{C_1} \ldots w_{C_p}$ is faithful to the evolution of $\mathcal{M}$ and, in particular, for each $1 \leq i < p$, $C_{i+1} = succ_l(C_i)$ if w_{C_i} is marked by l, and $C_{i+1} = succ_r(C_i)$ otherwise. By definition of the CTL^* formulas φ_{code} and φ_{good}, we can assume that w_ν is a code and T_{check} is a good check-tree associated with w_ν. Hence, for each block bl' of w_ν which does not belong to the first TM configuration, there is a path π_ν in T_{opt_3} such that the sequence of nodes associated with bl is marked by $check_1$. Moreover, this path visits exactly one opt_4-node and the following holds: the label of this node contains *end* and the associated block bl has the same block number as bl' and belongs to the TM configuration that precedes the TM configuration of bl along w_ν. Since a path π_ν of T_{opt_3} visits an opt_4-node iff the $check_1$-block of π_ν does not belong to the first TM configuration of w_ν, in order to ensure that w_ν is faithful to the evolution of $\mathcal{M}$, it suffices to require the following: for each path π_ν of T_{opt_3} containing an opt_4-node, it holds that $u' = next_d(u_p, u, u_s)$, where u' is the block content of the $check_1$-block bl' of π_ν, u is the cell content of the opt_4-block bl of π_ν, $d = l$ iff the TM configuration associated with bl' is marked by l, and u_s (resp., u_p) is the block content of the TM block—if any—that precedes (resp., follows) bl along π_ν and belongs to the same TM configuration as bl.

Therefore, $\varphi_{fair} := \varphi_{fi} \wedge \varphi_{la} \wedge \varphi_{non\text{-}ext}$, where φ_{fi} manages the case in which bl' is the first block of the associated TM configuration, φ_{la} manages the case in which bl' is the last TM block, and $\varphi_{non\text{-}ext}$ manages the remaining cases. For simplicity, we define only $\varphi_{non\text{-}ext}$ (the other two formulas can be defined similarly). Such a formula is defined as follows where $\widetilde{\Sigma} = \Sigma \cup (Q \times \Sigma)$ (recall that we can assume that φ_{fair} is asserted at the root of T_{opt_3}):

$$AG\Big(\big[b \wedge check_1 \wedge (Xend) \wedge F(b \wedge Xopt_4)\big] \longrightarrow \bigvee_{u_p,u,u_s \in \widetilde{\Sigma}} \bigvee_{d \in \{l,r\}} \big[next_d(u_p,u,u_s) \wedge F(d \wedge F(u_s \wedge X(opt_4 \wedge (\neg b\,\mathcal{U}\,(u \wedge X(\neg b\,\mathcal{U}\,u_p))))))\big]\Big) \qquad \square$$

Now, we can prove the main result of this paper.

Theorem 4

(1) *The pushdown module-checking problem for* CTL *is* 2EXPTIME-*complete.*
(2) *The pushdown module-checking problem for* CTL* *is* 3EXPTIME-*complete.*
(3) *The pushdown module-checking problem for both* CTL *and* CTL* *is* EXPTIME-*complete in the size of the given* OPD.

Proof Claims 1 and 2 directly follow from Theorems 1, 2, and 3. Now, let us consider Claim 3. First, we note that model checking pushdown systems corresponds to module checking the class of *OPD* in which there are not environment configurations. Moreover, pushdown model checking against *CTL* is known to be EXPTIME-complete also for a fixed formula [3]. Thus, Claim 3 follows from Theorem 1. □

6 Conclusion

Kupferman, Vardi, and Wolper [18] introduced module checking as a useful framework for the verification of open finite-state systems. There, it has been shown that while for *LTL* the complexity of the model checking problem coincides with that of module checking (i.e., it is PSPACE-complete), for the branching time paradigm the problem of module checking is much harder. In fact, *CTL* (resp., *CTL**) module checking of finite-state systems is EXPTIME-complete (resp. 2EXPTIME-complete).

In this paper, we have extended the framework of module checking problem to pushdown systems. Figure 4 below summarizes our results on pushdown module checking and compares them with those known on pushdown model checking. All the complexities in the figure denote tight bounds. Our complexities results provide an additional evidence that for pushdown systems, checking *CTL* or *CTL** properties is actually harder than checking *LTL* properties.

We conclude with some questions which are left open in this paper. The first interesting question is related to synthesis issues, i.e. whether it is possible to compute in case the module $M_{\mathcal{S}}$ associated with the given *OPD* $\mathcal{S}$ does not satisfy the given formula, a counterexample, i.e., a representation in some formalism of an environment (a labeled tree in $exec(M_{\mathcal{S}})$) which violates the specification. And in particular, is there a finite-state environment that is a witness for violating the property? If not, is there a pushdown environment

	Model Checking	System complexity of Model Checking	Module Checking	System complexity of Module Checking
LTL	EXPTIME [2]	PTIME [2]	EXPTIME	PTIME
CTL	EXPTIME [27]	EXPTIME [3]	2EXPTIME	EXPTIME
*CTL**	2EXPTIME [3]	EXPTIME [3]	3EXPTIME	EXPTIME

Fig. 4 Complexity results on pushdown module checking and pushdown model checking

that violates it, and in this case, is it possible to construct a pushdown system whose computation tree corresponds to one of these environments? For finite-state module checking, for example, if the module does not satisfy the formula, then it is guaranteed the existence of a finite-state environment which violates the specification, and in particular, one can construct a labeled finite-state graph whose unwinding corresponds to some of such finite-state environments.[11]

An other interesting question is as follows. It is well-known that the set of configurations of a pushdown system satisfying a *CTL* or *CTL** formula is regular and can be effectively computed (see for example [2, 12]). It would be interesting to investigate the same question in the context of pushdown module checking. Moreover, in this paper, the partition into system and environment configurations of an *OPD* is only based on the control state and stack top symbol. The more general situation in which the partition depends also on the whole stack content seems interesting, and in particular, the case in which the set of environment or system configurations is symbolically represented by a regular specification (it is easy to check that with context-free specifications, the problems become undecidable).

Finally, since the conference publication of our paper [4], further questions involving module checking of pushdown systems have been addressed [1, 13]. In particular, pushdown module checking against branching-time temporal logics more expressive than *CTL** has been addressed in [13], where the authors show that for the μ-calculus enriched with graded and nominals (hybrid graded μ-calculus), the problem is still decidable and is solvable in double exponential time in the size of the formula and in single exponential time in the size of the system. Aminof, Murano, and Vardi [1] extend the pushdown module checking framework to the imperfect information setting for the case in which the environment has only a partial view of the system's control states and stack content. It has been shown that *CTL* pushdown module checking becomes undecidable when the imperfect information relies on the pushdown store content, while it is decidable and its complexity is the same as that of (perfect information) pushdown module checking when the imperfect information relies only on the control states.

References

1. Aminof A, Murano A, Vardi MY (2007) Pushdown module checking with imperfect information. In: Proc 18th international conference on concurrency theory (CONCUR'07). LNCS, vol 4703. Springer, Berlin, pp 461–476

[11] This is a consequence of the fact that a parity *NTA* accepts some (labeled) tree iff it accepts some regular tree, i.e., a tree with a finite number of distinct subtrees.

2. Bouajjani A, Esparza J, Maler O (1997) Reachability analysis of pushdown automata: application to model-checking. In: Proc 8th international conference on concurrency theory (CONCUR'97). LNCS, vol 1243. Springer, Berlin, pp 135–150
3. Bozzelli L (2006) Complexity results on branching-time pushdown model checking. In: Proc 7th conference on verification, model checking, and abstract interpretation (VMCAI'06). LNCS, vol 3855. Springer, Berlin, pp 65–79
4. Bozzelli L, Murano A, Peron A (2005) Pushdown module checking. In: Proc 12th int conf on logic for programming, artificial intelligence, and reasoning (LPAR'05). LNCS, vol 3835. Springer, Berlin, pp 504–518
5. Buchi JR (1962) On a decision method in restricted second order arithmetic. In: Proc internat congr logic, method and philos sci 1960, Stanford, pp 1–12
6. Cachat T (2002) Two-way tree automata solving pushdown games. In: Automata, logics, and infinite games. LNCS, vol 2500. Springer, Berlin, pp 303–317
7. Chandra AK, Kozen DC, Stockmeyer LJ (1981) Alternation. J ACM 28(1):114–133
8. Clarke EM, Emerson EA (1981) Design and verification of synchronization skeletons using branching time temporal logic. In: Proceedings of workshop on logic of programs. LNCS, vol 131. Springer, Berlin, pp 52–71
9. Emerson EA, Halpern JY (1986) Sometimes and not never revisited: on branching versus linear time. J ACM 33(1):151–178
10. Emerson EA, Jutla CS (1988) The complexity of tree automata and logics of programs. In: 29th annual IEEE symposium on foundations of computer science (FOCS'88), pp 328–337
11. Emerson EA, Jutla CS (1991) Tree automata, μ-calculus and determinacy. In: 32nd annual IEEE symposium on the foundations of computer science (FOCS'91), pp 368–377
12. Esparza J, Kucera A, Schwoon S (2003) Model checking LTL with regular valuations for pushdown systems. Inf Comput 186(2):355–376
13. Ferrante A, Murano A, Parente M (2008) Enriched μ-calculi module checking. Log Methods Comput Sci 4(3):1–21
14. Hoare CAR (1985) Communicating sequential processes. Prentice-Hall, New York
15. Kupferman O, Grumberg O (1996) Buy one, get one free!!! J Log Comput 6(4):523–539
16. Kupferman O, Thiagarajan PS, Madhusudan P, Vardi MY (2000) Open systems in reactive environments: Control and Synthesis. In: Proc 11th international conference on concurrency theory (CONCUR'00). LNCS, vol 1877. Springer, Berlin, pp 92–107
17. Kupferman O, Vardi MY, Wolper P (2000) An automata-theoretic approach to branching-time model checking. J ACM 47(2):312–360
18. Kupferman O, Vardi MY, Wolper P (2001) Module checking. Inf Comput 164(2):322–344
19. Kupferman O, Piterman N, Vardi MY (2002) Pushdown specifications. In: 9th int conf on logic for programming, artificial intelligence, and reasoning (LPAR'02). LNAI, vol 2514. Springer, Berlin, pp 262–277
20. Loding C, Madhusudan P, Serre O (2004) Visibly pushdown games. In: Proc 24th conference on foundations of software technology and theoretical computer science (FST&TCS'04). Springer, Berlin, pp 408–420
21. Miyano S, Hayashi T (1984) Alternating finite automata on ω-words. Theor Comput Sci 32:321–330
22. Muller DE, Shupp PE (1985) The theory of ends, pushdown automata, and second-order logic. Theor Comput Sci 37:51–75
23. Queille JP, Sifakis J (1981) Specification and verification of concurrent programs in Cesar. In: Proceedings of the fifth international symposium on programming. LNCS, vol 137. Springer, Berlin, pp 337–351
24. Vardi MY (1998) Reasoning about the past with two-way automata. In: Proc 25th international colloquium on automata, languages and programming (ICALP'98). LNCS, vol 1443. Springer, Berlin, pp 628–641
25. Vardi MY, Wolper P (1986) Automata-theoretic techniques for modal logics of programs. J Comput Syst Sci 32(2):182–221
26. Walukiewicz I (1996) Pushdown processes: games and model checking. In: Proc 8th international conference on computer aided verification (CAV'96). LNCS, vol 1102. Springer, Berlin, pp 62–74
27. Walukiewicz I (2000) Model checking CTL properties of pushdown systems. In: Proc 20th conference on foundations of software technology and theoretical computer science (FST&TCS'00). LNCS, vol 1974. Springer, Berlin, pp 127–138
28. Walukiewicz I (2002) Monadic second-order logic on tree-like structures. Theor Comput Sci 275:311–346