Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 379 (2007) 286–297
Theoretical Computer Science
www.elsevier.com/locate/tcs

Complexity results on branching-time pushdown model checking[☆]

Laura Bozzelli[*]

LSV, CNRS & ENS Cachan - 61, avenue du President Wilson - 94235 CACHAN Cedex, France

Received 24 April 2006; received in revised form 21 March 2007; accepted 24 March 2007

Communicated by J. Esparza

Abstract

The model checking problem of pushdown systems (PMC problem, for short) against standard branching temporal logics has been intensively studied in the literature. In particular, for the modal μ-calculus, the most powerful branching temporal logic used for verification, the problem is known to be EXPTIME-complete (even for a fixed formula). The problem remains EXPTIME-complete also for the logic *CTL*, which corresponds to a fragment of the alternation-free modal μ-calculus. For the logic *CTL**, the problem is known to be in 2EXPTIME. In this paper, we show that the complexity of the PMC problem for *CTL** is in fact 2EXPTIME-complete. Moreover, we give a new optimal algorithm to solve this problem based on automata theoretic techniques. Finally, we prove that the program complexity of the PMC problem against *CTL* (i.e., the complexity of the problem in terms of the size of the system) is EXPTIME-complete.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Infinite-state systems; Pushdown systems; Model checking; Regular propositional temporal logics

1. Introduction

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 given by a formula in a suitable propositional temporal logic. There are two types of temporal logic: linear and branching. In linear temporal logics, each moment in time has a unique possible future (formulas are interpreted over linear sequences corresponding to single computations of the system), while in branching temporal logics, each moment in time may split into several possible futures (formulas are interpreted over infinite trees, which describe all the possible computations of the system). The size of an instance of a model checking problem depends on two parameters: the size of the finite formal description of the given system and the size of the formula. In practice, the formula is normally very small, while the description of the system is often very large. Therefore, the complexity of the problem in terms of the size of the system (called *program complexity*) is very important in practice. Traditionally, model checking is applied to finite-state systems, typically modelled by labelled state-transition graphs.

[☆] A preliminary version of this paper appears in the Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'06, LNCS 3855, pages 65–79. Springer, 2006.

[*] Corresponding address: Universita di Napoli, Dipartimento di Matematica e Applicazioni, Via Cintia, 80126 Napoli, Italy. Tel.: +39 081 662316; fax: +39 081 7145110.
E-mail addresses: bozzelli@lsv.ens-cachan.fr, laura.bozzelli@dma.unina.it.

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.03.049

Recently, the investigation of model-checking techniques has been extended to infinite-state systems. An active field of research is the model-checking of 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 concerning the decidability of the monadic second-order theory of *context-free systems* [17]. This result can be extended to *pushdown systems* [3], and it implies decidability of the model checking problem for all those logics (modal μ-calculus, *CTL**, *CTL*, etc.) which have effective translations to the monadic second-order logic. As this general decidability result gives a non-elementary upper bound for the complexity of model checking, researchers have sought decidability results of elementary complexity. Concerning pushdown systems, model checking with branching-time logics is quite hard. In particular, Walukiewicz [24] has shown that model checking these systems with respect to modal μ-calculus, the most powerful branching temporal logic used for verification, 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* [25], which corresponds to a fragment of the alternation-free modal μ-calculus. However, the exact complexity in the size of the system (for a fixed *CTL* formula) is an open problem: it lies somewhere between PSPACE and EXPTIME [1]. In [25], Walukiewicz has shown that even for the simple branching-time logic EF (a fragment of *CTL*), the problem is quite hard since it is PSPACE-complete (even for a fixed formula). For the branching-time temporal logic *CTL**, Esparza et al. in [12] have shown that the problem is in 2EXPTIME by an exponential time reduction (in the size of the formula) to the pushdown model checking problem against *LTL*. However, the exact complexity of the problem is unknown.

For standard linear temporal logics, model-checking pushdown systems with *LTL* and the *linear-time μ-calculus* are EXPTIME-complete [1]. However, the problem is polynomial in the size of the pushdown system. It follows that the problem is only slightly harder than for finite-state systems, where it is PSPACE-complete but polynomial for any fixed formula [20,21]. For optimal pushdown model-checking algorithms, see also [2,13,11,12,19].

In this paper, we study the complexity of the pushdown model checking problem (PMC problem, for short) against *CTL** and the program complexity of the PMC problem against *CTL*. In particular, we prove the following two results:

- The PMC problem against *CTL** is 2EXPTIME-complete (and EXPTIME-complete in the size of the system).
- The program complexity of the PMC problem against *CTL* is EXPTIME-complete.

Note that for finite-state systems, the model checking problem for *CTL** has the same complexity as the model checking for *LTL* [6], and it can be solved by combining the state labelling technique from *CTL* model checking (based on a sequence of reachability tests) with *LTL* model checking. Moreover, for pushdown systems, as seen earlier, the model checking problem for both *CTL* and *LTL* is EXPTIME-complete. This would suggest that the PMC problem for *CTL** may be solvable in single exponential time. Therefore, our 2EXPTIME-hardness result could seem slightly surprising. However, we also note a correlation in terms of complexity between the branching-time PMC problem and the satisfiability problem for branching-time temporal logics. Indeed, it is well known that the satisfiability problem is EXPTIME-complete for *CTL* [7] and modal μ-calculus [9], and it is 2EXPTIME-complete for *CTL** [9,23].

As mentioned above, for the PMC problem against *CTL**, membership in 2EXPTIME was shown in [12]. Here, we give a new optimal algorithm to solve this problem which is based on automata-theoretic techniques. In particular, we propose an exponential time reduction (in the size of the formula) to the emptiness problem of *alphabet-free alternating parity pushdown automata*. The emptiness problem for this class of automata can be solved by a construction similar to that given in [14] to solve the emptiness problem for nondeterministic parity pushdown tree automata (the algorithm in [14] is based on a polynomial reduction to the emptiness of two-way alternating parity finite-state tree automata, which is known to be decidable in exponential time [22]). 2EXPTIME-hardness is shown by a technically non-trivial reduction from the word problem for EXPSPACE-bounded alternating Turing Machines.

The EXPTIME-hardness of the pushdown model checking problem against *CTL* was shown by Walukiewicz [25] using a reduction from the word problem for PSPACE-bounded alternating Turing Machines. We use the basic ideas of the construction in [25] in order to prove that the program complexity of the problem (i.e., assuming the *CTL* formula is fixed) is still EXPTIME-hard.

2. Preliminaries

In this section we recall syntax and semantics of *CTL** and *CTL* [8,5]. Also, we define pushdown systems and the model checking problem.

CTL* and CTL logics. The logic *CTL** is a branching-time temporal logic [8], 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, 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 by the following syntax:

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

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 *Computation Tree Logic CTL* [5] is a restricted subset of *CTL**, obtained by 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.

The models for the logic *CTL** are labelled graphs $\langle W, \rightarrow, \mu\rangle$, where W is a countable set of vertices, $\rightarrow \subseteq W \times W$ is the edge relation, and $\mu : W \rightarrow 2^{AP}$ maps each vertex $w \in W$ to the set of atomic propositions that hold in w. Such labelled graphs are called transition systems (TS, for short) here. In this context, vertices are also called (global) states. For $w \rightarrow w'$, we say that w' is a successor of w. A path is a (finite or infinite) sequence of vertices $\pi = w_0, w_1, \ldots$ such that $w_{i-1} \rightarrow w_i$ for every $1 \leq i < |\pi|$. We denote the suffix $w_i, w_{i+1}, \ldots$ of π by π^i, and the i-th vertex of π by $\pi(i)$. A *maximal* path is either an infinite path or a finite path leading to a vertex without successors.

Let $G = \langle W, \rightarrow, \mu\rangle$ be an TS, $w \in W$, and π be a maximal path of G. For a state (resp., path) formula φ (resp., θ), the satisfaction relation $(G, w) \models \varphi$ (resp., $(G, \pi) \models \theta$), meaning that φ (resp., θ) holds at state w (resp., holds along π) in G, is defined by induction. The clauses for proposition letters, negation, and conjunction are standard. For the other constructs we have:

- $(G, w) \models A\,\theta$ *iff* for each maximal path π in G from w, $(G, \pi) \models \theta$;
- $(G, w) \models E\,\theta$ *iff* there exists a maximal path π from w such that $(G, \pi) \models \theta$;
- $(G, \pi) \models \varphi$ *iff* $(G, \pi(0)) \models \varphi$;
- $(G, \pi) \models X\theta$ *iff* $\pi(1)$ is defined and $(G, \pi^1) \models \theta$;
- $(G, \pi) \models \theta_1\,\mathcal{U}\,\theta_2$ *iff* there exists $i \geq 0$ such that $(G, \pi^i) \models \theta_2$ and for all $0 \leq j < i$, we have $(G, \pi^j) \models \theta_1$.

Pushdown systems. A pushdown system (*PDS*, for short) is a tuple $\mathcal{S} = \langle AP, \Gamma, P, \Delta, L\rangle$, where AP is a finite set of proposition names, Γ is a finite stack alphabet, P is a finite set of (control) states, $\Delta \subseteq (P \times (\Gamma \cup \{\gamma_0\})) \times (P \times \Gamma^*)$ is a *finite* set of transition rules (where $\gamma_0 \notin \Gamma$ is the *stack bottom symbol*), and $L : P \times (\Gamma \cup \{\gamma_0\}) \rightarrow 2^{AP}$ is a labelling function. 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. For each $(p, B) \in P \times (\Gamma \cup \{\gamma_0\})$, we denote by $next_{\mathcal{S}}(p, B)$ the finite set (possibly empty) of the pairs (p', β) such that $\langle (p, B), (p', \beta)\rangle \in \Delta$. The size $|\mathcal{S}|$ of $\mathcal{S}$ is $|P| + |\Delta|$, with $|\Delta| = \sum_{\langle (p,B),(p',\beta)\rangle \in \Delta} |\beta|$.

The semantics of an *PDS* $\mathcal{S} = \langle AP, \Gamma, P, \Delta, L\rangle$ is described by a TS $G_{\mathcal{S}} = \langle W, \rightarrow, \mu\rangle$, where W is the set of pushdown configurations, for all $(p, B \cdot \alpha) \in W$ with $B \in \Gamma \cup \{\gamma_0\}$, $\mu(p, B \cdot \alpha) = L(p, B)$, and $\rightarrow$ is defined as follows:

- $(p, B \cdot \alpha) \rightarrow (p', \beta)$ *iff* there is $\langle (p, B), (p', \beta')\rangle \in \Delta$ such that either $B \in \Gamma$ and $\beta = \beta' \cdot \alpha$, or $B = \gamma_0$ (note that α is empty) and $\beta = \beta' \cdot \gamma_0$ (note that every transition that removes the bottom symbol γ_0 also pushes it back);

The *pushdown model checking* problem (PMC problem, for short) against *CTL* (resp., *CTL**) is to decide, for a given *PDS* $\mathcal{S}$, an initial configuration w_0 of $\mathcal{S}$, and a *CTL* (resp., *CTL**) formula φ, whether $(G_{\mathcal{S}}, w_0) \models \varphi$.

3. Tree automata

In order to solve the PMC problem for *CTL**, we use an automata theoretic approach; in particular, we exploit the formalisms of *Alternating Parity* (*finite-state*) *Tree automata* (*APT*, for short) [18,10] and *Alphabet-free alternating parity pushdown automata* (*PD-APA*, for short).

Let $\mathbb{N}$ be the set of positive integers. A *tree* T is a subset of $\mathbb{N}^*$ such that if $i \cdot x \in T$ for some $i \in \mathbb{N}$ and $x \in \mathbb{N}^*$, then also $x \in T$ and for all $1 \leq j < i$, $j \cdot x \in T$. 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* (or *successors*) of x (in T) is $children(T, x) = \{i \cdot x \in T \mid i \in \mathbb{N}\}$. 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 $k \geq 1$, the (complete) k-ary tree is the tree $\{1, \ldots, k\}^*$. For an alphabet Σ, a Σ-labelled 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. Note that $\langle T, V\rangle$ corresponds to the labelled graph $G_T = \langle T, \to, V\rangle$ where $x \to y$ iff $y \in children(T, x)$. If $\Sigma = 2^{AP}$, then for every *CTL** formula φ over AP, we say that $\langle T, V\rangle$ satisfies φ if $(G_T, \varepsilon) \models \varphi$.

For a set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X. Elements of X are called *atoms*. For $Y \subseteq X$ and $\psi \in \mathcal{B}^+(X)$, we say that Y satisfies ψ iff assigning **true** to all elements of Y and assigning **false** to all elements of $X \setminus Y$, makes ψ true. For $k \geq 1$, we denote by $[k]$ the set $\{1, \ldots, k\}$.

Alternating parity (finite-state) tree automata (*APT*). We describe *APT* over (complete) k-ary trees for a given $k \geq 1$. Formally, an *APT* 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, $\delta : Q \times \Sigma \to \mathcal{B}^+([k] \times Q)$ is a transition function, and F is a parity acceptance condition [10], i.e., $F = \{F_1, \ldots, F_m\}$ is a sequence of subsets of Q, where $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_m = Q$ (m is called the index of $\mathcal{A}$).

A run of $\mathcal{A}$ on a Σ-labelled k-ary tree $\langle T, V\rangle$ (where $T = [k]^*$) is a labelled tree $\langle T_r, r\rangle$ in which each node is labelled by an element of $T \times Q$. A node in T_r labelled by (x, q) describes a copy of the automaton that is in the state q and reads the node x of T. Note that many nodes of T_r can correspond to the same node of T. The labels of a node and its children (successors) have to satisfy the transition function. Formally, a run over $\langle T, V\rangle$ is a $T \times Q$-labelled tree $\langle T_r, r\rangle$ such that $r(\varepsilon) = (\varepsilon, q_0)$, and for all $y \in T_r$ with $r(y) = (x, q)$, the following holds:

- there is a (possibly empty) set $\{(h_1, q_1), \ldots, (h_n, q_n)\} \subseteq [k] \times Q$ *satisfying* $\delta(q, V(x))$ such that for each $1 \leq j \leq n$, $j \cdot y \in T_r$ and $r(j \cdot y) = (h_j \cdot x, q_j)$.

Note that several copies of the automaton may go in the same direction (in the input tree) and that the automaton is not required to send copies to all the directions. The automaton $\mathcal{A}$ is *balanced* if for each $(q, \sigma) \in Q \times \Sigma$, $\delta(q, \sigma)$ is a positive Boolean combination of sub-formulas (called *generators*) either of the form $\bigvee_{i=1}^{i=k}(i, q')$ or of the form $\bigwedge_{i=1}^{i=k}(i, q')$ (note that q' is independent from the specific direction i). The *size* $|\mathcal{A}|$ of a balanced *APT* $\mathcal{A}$ is $|Q| + |\delta| + |F|$, where $|\delta| = \sum_{(q,\sigma)\in Q\times\Sigma} |\delta(q, \sigma)|$ and $|\delta(q, \sigma)|$ is the length of the formula obtained from $\delta(q, \sigma)$ considering each generator occurring in $\delta(q, \sigma)$ as an atomic proposition.

For a run $\langle T_r, r\rangle$ over $\langle T, V\rangle$ and an infinite path $\pi \subseteq T_r$, let $inf_r(\pi) \subseteq Q$ be the set such that $q \in inf_r(\pi)$ iff there are infinitely many $y \in \pi$ such that $r(y) \in T \times \{q\}$. For the 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 $1 \leq j < i$, $inf_r(\pi) \cap F_j = \emptyset$. A run $\langle T_r, r\rangle$ is *accepting* if all its infinite 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-labelled (complete) k-ary trees accepted by $\mathcal{A}$.

It is well known that formulas of *CTL** can be translated to tree automata. In particular, we are interested in optimal translations to balanced *APT*.

Lemma 1 (*[16]*). *Given a CTL* formula φ over AP and $k \geq 1$, we can construct a* balanced *APT of size $O(2^{|\varphi|})$ and index $O(|\varphi|)$ that accepts exactly the set of 2^{AP}-labelled complete k-ary trees satisfying φ.*[1]

Alphabet-free alternating parity pushdown automata (*PD-APA*). An *PD-APA* is a tuple $\mathcal{P} = \langle \Gamma, P, p_0, \alpha_0, \rho, F\rangle$, where Γ is a finite stack alphabet, P is a finite set of (control) states, $p_0 \in P$ is an initial state, $\alpha_0 \in \Gamma^* \cdot \gamma_0$ is an initial stack content, $\rho : P \times (\Gamma \cup \{\gamma_0\}) \to \mathcal{B}^+(P \times \Gamma^*)$ is a transition function, and $F = \{F_1, \ldots, F_m\}$ is a *parity* acceptance condition over P. Intuitively, when the automaton $\mathcal{P}$ is in state p and the stack contains a word $B \cdot \alpha \in \Gamma^*.\gamma_0$, then $\mathcal{P}$ chooses a (possibly empty) finite set $\{(p_1, \beta_1), \ldots, (p_n, \beta_n)\} \subseteq P \times \Gamma^*$ satisfying $\rho(p, B)$, and splits into n copies such that for each $1 \leq j \leq n$, the j-th copy moves to state p_j and updates the stack content by removing B and pushing β_j.
Formally, a run of $\mathcal{P}$ is a $P \times \Gamma^*.\gamma_0$-labelled tree $\langle T_r, r\rangle$ such that $r(\varepsilon) = (p_0, \alpha_0)$ and for all $y \in T_r$ with $r(y) = (p, B \cdot \alpha)$ and $B \in \Gamma \cup \{\gamma_0\}$, the following holds:

[1] Ref. [16] gives a translation from *CTL** to *Hesitant alternating tree automata* which are a special case of parity alternating tree automata.

- there is a (possibly empty) finite set $\{(p_1, \beta_1), \dots, (p_n, \beta_n)\} \subseteq P \times \Gamma^*$ *satisfying* $\rho(p, B)$ such that for each $1 \leq j \leq n$, $j \cdot y \in T_r$ and $r(j \cdot y) = (p_j, \beta_j \cdot \alpha)$ if $B \neq \gamma_0$, and $r(j \cdot y) = (p_j, \beta_j \cdot \gamma_0)$ otherwise (note that in this case $\alpha = \varepsilon$).

The notion of *accepting* path $\pi \subseteq T_r$ is defined as for *APT* with $inf_r(\pi)$ defined as follows: $inf_r(\pi) \subseteq P$ is the set such that $p \in inf_r(\pi)$ iff there are infinitely many $y \in \pi$ for which $r(y) \in \{p\} \times \Gamma^* \cdot \gamma_0$. A run $\langle T_r, r\rangle$ is *accepting* if every infinite path $\pi \subseteq T_r$ is accepting. The *emptiness* problem for *PD-APA* is to decide, for a given *PD-APA*, the existence of an accepting run.

For $(p, \alpha) \in P \times \Gamma^*$, the size of (p, α) is $|\alpha|$. The size $|\rho|$ of the transition function is given by $\sum_{(p,B) \in P \times (\Gamma \cup \{\gamma_0\})} |\rho(p, B)|$, where $|\rho(p, B)|$ is the sum of the sizes of the occurrences of atoms in $\rho(p, B)$.

In the following, we are interested in the emptiness problem for *PD-APA*. In [14], an optimal algorithm is given to solve the emptiness problem for nondeterministic parity pushdown tree automata. This algorithm is based on a polynomial reduction to the emptiness of two-way alternating parity tree automata, which is known to be decidable in exponential time [22]. By a straightforward readaptation of the proof given in [14] we obtain the following.

Proposition 1. *The emptiness problem for PD-APA with index m and transition function ρ is solvable in time exponential in $m \cdot |\rho|$.*

4. An automata-theoretic algorithm for the PMC problem against *CTL**

In this section, we give an automata-theoretic algorithm to solve the PMC problem against *CTL**. We fix a *PDS* $\mathcal{S} = \langle AP, \Gamma, P, \Delta, L\rangle$, an initial configuration $w_0 = (p_0, \alpha_0)$ of $\mathcal{S}$, and a *CTL** formula φ. The unwinding of the TS $G_{\mathcal{S}} = \langle W, \rightarrow, \mu\rangle$ from w_0 induces a W-labelled tree $\langle T_{\mathcal{S}}, V_{\mathcal{S}}\rangle$: the root of $T_{\mathcal{S}}$ is associated with the initial configuration w_0, and the children of each node $x \in T_{\mathcal{S}}$ labelled by $w \in W$ correspond to the successors of w.[2] In the following, we sometime view $\langle T_{\mathcal{S}}, V_{\mathcal{S}}\rangle$ as a 2^{AP}-labelled tree, taking the label of a node x to be $\mu(V_{\mathcal{S}}(x))$ instead of $V_{\mathcal{S}}(x)$. Which interpretation is intended will be clear from the context.

Evidently, $(G_{\mathcal{S}}, w_0) \models \varphi$ iff $\langle T_{\mathcal{S}}, V_{\mathcal{S}}\rangle$ satisfies φ. Therefore, the model checking problem of $\mathcal{S}$ against φ can be reduced to check whether $\langle T_{\mathcal{S}}, V_{\mathcal{S}}\rangle$ belongs to the language of the *APT* (whose existence is guaranteed by Lemma 1) accepting the tree-models of φ. However, the branching degree of $T_{\mathcal{S}}$ is not uniform and, in particular, some nodes of $T_{\mathcal{S}}$ may not have successors. We solve this problem as follows. Note that the number of successors of each configuration is finite. Moreover, the maximum of such numbers, denoted by k, is defined and can be trivially computed from the transition relation Δ of $\mathcal{S}$. We can encode the computation tree $\langle T_{\mathcal{S}}, V_{\mathcal{S}}\rangle$ as a $2^{AP \cup \{t\}} \cup \{nil\}$-labelled complete k-ary tree (where *nil* 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 configurations without successors) of the tree $T_{\mathcal{S}}$; second, for each node $x \in T_{\mathcal{S}}$ with d children $1 \cdot x, \dots, d \cdot x$ (note that $0 \leq d \leq k$), we add the children $(d + 1) \cdot x, \dots, k \cdot x$ and label these new nodes with *nil*; finally, for each node x labelled by *nil* we add recursively k children labelled by *nil*. Let $\langle [k]^*, \widetilde{V}_{\mathcal{S}}\rangle$ be the tree thus obtained. Since a node labelled by *nil* stands for a node that actually does not exist, we have to take this into account when we interpret *CTL** formulas over the tree $\langle [k]^*, \widetilde{V}_{\mathcal{S}}\rangle$. This means that we have to consider only the paths in this tree (called "legal" paths) that either never visit a node labelled by *nil* or contain a *terminal* node (i.e. a node labelled by t). Note that a path is *not* "legal" iff it satisfies the formula $\neg t\, \mathcal{U}\, nil$. In order to achieve this, we define inductively a function f : *CTL** formulas $\rightarrow$ *CTL** formulas[3] such that $f(\varphi)$ restricts path quantification to only "legal" paths:

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

[2] Assuming that W is ordered, there is indeed only a single such tree. Since *CTL** formulas cannot distinguish between trees obtained by different orders, we do not lose generality by considering a particular order.

[3] Here, for *CTL** formulas we mean both state and path formulas.

Note that $|f(\varphi)| = O(|\varphi|)$. By definition of f, it follows that $\langle T_{\mathcal{S}}, V_{\mathcal{S}}\rangle$ satisfies φ (i.e., $(G_{\mathcal{S}}, w_0) \models \varphi$) *iff* $\langle [k]^*, \widetilde{V}_{\mathcal{S}}\rangle$ satisfies $f(\varphi)$.

Let $\mathcal{A}_{f(\varphi)} = \langle 2^{AP\cup\{t\}} \cup \{nil\}, Q, q_0, \delta, F\rangle$, with $F = \{F_1, \ldots, F_m\}$, be the balanced *APT* (whose existence is guaranteed by Lemma 1) accepting exactly the $2^{AP\cup\{t\}} \cup \{nil\}$-labelled complete k-ary trees that satisfy $f(\varphi)$. We have to check whether $\langle [k]^*, \widetilde{V}_{\mathcal{S}}\rangle$ belongs to the language $\mathcal{L}(\mathcal{A}_{f(\varphi)})$. We reduce this problem to the emptiness of a *PD-APA* $\mathcal{P} = \langle \Gamma, (P \cup \{nil\}) \times Q, (p_0, q_0), \alpha_0, \rho, F'\rangle$, which is defined as follows. The states of $\mathcal{P}$ consist either of pairs of states of $\mathcal{S}$ and states of $\mathcal{A}_{f(\varphi)}$, or pairs of the form (nil, q) where q is a state of $\mathcal{A}_{f(\varphi)}$. Intuitively, when the automaton $\mathcal{P}$ is in state $(p, q) \in P \times Q$ with stack content α, and (p, α) is a configuration associated with some node x of $\langle T_{\mathcal{S}}, V_{\mathcal{S}}\rangle$, then $\mathcal{P}$ simulates the behaviour of $\mathcal{A}_{f(\varphi)}$ starting from state q on the input tree given by the subtree of $\langle [k]^*, \widetilde{V}_{\mathcal{S}}\rangle$ rooted at node x. Moreover, in state (nil, q), $\mathcal{P}$ simulates the behaviour of $\mathcal{A}_{f(\varphi)}$ from state q on the input tree in which all nodes are labelled by *nil*.

The parity acceptance condition F' is given by $\{(P \cup \{nil\}) \times F_1, \ldots, (P \cup \{nil\}) \times F_m\}$. Finally, the transition function ρ is defined as follows:

- for each $(p, q) \in P \times Q$ and $B \in \Gamma \cup \{\gamma_0\}$, $\rho((p, q), B)$ is defined as follows. Let $next_{\mathcal{S}}(p, B) = \{(p_1, \alpha_1), \ldots, (p_d, \alpha_d)\}$ (note that $0 \le d \le k$). If $d > 0$ (resp., $d = 0$), then $\rho((p, q), B)$ is obtained from formula $\delta(q, L(p, B))$ (resp., $\delta(q, L(p, B) \cup \{t\})$) by replacing each generator occurring in it of the form $\bigvee_{i=1}^{i=k}(i, q')$ with $\bigvee_{i=1}^{i=d}((p_i, q'), \alpha_i) \vee ((nil, q'), \varepsilon)$, and each generator of the form $\bigwedge_{i=1}^{i=k}(i, q')$ with $\bigwedge_{i=1}^{i=d}((p_i, q'), \alpha_i) \wedge ((nil, q'), \varepsilon)$;
- for each $q \in Q$ and $B \in \Gamma \cup \{\gamma_0\}$, $\rho((nil, q), B)$ is obtained from formula $\delta(q, nil)$ by replacing each generator occurring in it of the form $\mathcal{C}_{i=1}^{i=k}(i, q')$, where $\mathcal{C} \in \{\bigvee, \bigwedge\}$, with $((nil, q'), \varepsilon)$.

By construction, it easily follows that $\mathcal{P}$ has an accepting run iff $\langle [k]^*, \widetilde{V}_{\mathcal{S}}\rangle \in \mathcal{L}(\mathcal{A}_{f(\varphi)})$.

Note that the size $|\rho|$ of the transition function of $\mathcal{P}$ is bounded by $|\delta| \cdot |\Delta|$. By Lemma 1, it follows that $\mathcal{P}$ has index $O(|\varphi|)$ and $|\rho|$ is bounded by $O(2^{|\varphi|} \cdot \Delta)$. Then, by Proposition 1 we obtain the main result of this section.

Theorem 1. *Given a PDS $\mathcal{S} = \langle AP, \Gamma, P, \Delta, L\rangle$, a configuration w_0 of $\mathcal{S}$, and a CTL* formula φ, the model checking problem of $\mathcal{S}$ with respect to φ is solvable in time* exponential *in* $|\Delta| \cdot 2^{|\varphi|}$.

5. Lower bounds

In this section, we give lower bounds for the PMC problem against *CTL**, and for the program complexity of the PMC problem against *CTL*. The lower bound for *CTL* (resp., *CTL**) is shown by a reduction from the word problem for PSPACE-bounded (resp., EXPSPACE-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 $\delta : Q \times \Sigma \to (Q \times \Sigma \times \{L, R\}) \times (Q \times \Sigma \times \{L, R\})$.

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 belongs to F. Given an input $x \in \Sigma^*$, a computation tree of $\mathcal{M}$ on x is a tree in which each node corresponds to a configuration. The root of the tree corresponds to the initial configuration associated with x.[4] A node that corresponds to a universal configuration (i.e., the associated state is in $Q_\forall$) has two successors, corresponding to $succ_l(c)$ and $succ_r(c)$, while a node that corresponds to an existential configuration (i.e., the associated state is in $Q_\exists$) has a single successor, corresponding to either $succ_l(c)$ or $succ_r(c)$.

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

The tree is *accepting* if all its paths (from the root) visit an accepting configuration. An input $x \in \Sigma^*$ is *accepted* by $\mathcal{M}$ if there exists an accepting computation tree of $\mathcal{M}$ on x.

If $\mathcal{M}$ is PSPACE-bounded (resp., EXPSPACE-bounded), then there is a constant $k \geq 1$ such that for each $x \in \Sigma^*$, the space needed by $\mathcal{M}$ on input x is bounded by $k \cdot |x|$ (resp., $2^{k \cdot |x|}$). It is well known [4] that EXPTIME (resp., 2EXPTIME) coincides with the class of all languages accepted by PSPACE-bounded (resp., EXPSPACE-bounded) alternating Turing Machines.

The EXPTIME-hardness of the pushdown model checking problem against *CTL* was shown by Walukiewicz [25] using a reduction from the word problem for PSPACE-bounded alternating Turing Machines. We use the basic ideas of the construction in [25] in order to prove that the program complexity of the problem (i.e., assuming the *CTL* formula is fixed) is still EXPTIME-hard.

Theorem 2. *The program complexity of the PMC problem for CTL is* EXPTIME*-hard.*

Proof. We show that there is a *CTL* formula φ such that given a PSPACE-bounded alternating Turing Machine $\mathcal{M} = \langle \Sigma, Q, Q_\forall, Q_\exists, q_0, \delta, F \rangle$ and an input x, it is possible to define a *PDS* $\mathcal{S}$ and a configuration w of $\mathcal{S}$, whose sizes are *polynomial* in $n = k \cdot |x|$ and in $|\mathcal{M}|$,[5] such that $\mathcal{M}$ accepts x *iff* $(G_\mathcal{S}, w) \models \varphi$.

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

$\mathcal{S}$ guesses accepting computation trees of $\mathcal{M}$ starting from TM configurations of length n. The internal nodes of these trees are non-accepting configurations, and the leaves are accepting configurations. The trees are traversed as follows. If the current non-accepting configuration c is universal, then $\mathcal{S}$, first, will examine the subtrees associated with the left successor of c, and successively the subtrees associated with the right successor. If instead c is existential, then $\mathcal{S}$ will guess one of the two successors of c and, consequently, it will examine only the subtrees associated with this successor. In order to guess an accepting tree (if any) from a given configuration, $\mathcal{S}$ keeps track on the stack of the path from the root to the actual TM configuration by pushing the newly guessed configurations and popping when backtracking along the accepting subtree guessed so far. The stack alphabet of $\mathcal{S}$ is given by $\Sigma \cup (Q \times \Sigma) \cup \{\exists_l, \exists_r, \forall_l, \forall_r\}$ where $\exists_l$ and $\exists_r$ (resp., $\forall_l$ and $\forall_r$) are used to delimit the left and right successors of an existential (resp., universal) configuration. The behaviour of $\mathcal{S}$ can be subdivided in three steps.

1. *Generation of a TM configuration* (*operative phase*) — $\mathcal{S}$ generates nondeterministically, by push transitions, a TM configuration c followed by a symbol in $\{\forall_l, \exists_l, \exists_r\}$ on the stack, with the constraint that $\forall_l$ is chosen iff c is a universal configuration (i.e., the TM state q associated with c belongs to $Q_\forall$). In this phase, a (control) state of $\mathcal{S}$ has the form $(gen, q, i, flag)$, where $q \in Q$ keeps track of the TM state associated with c, gen is a label identifying the current operation of $\mathcal{S}$, $i \in \{0, \dots, n+1\}$ is used to ensure that c has exactly length n, and $flag \in \{0, 1\}$ is used to ensure that $c \in \Sigma^* \cdot (Q \times \Sigma) \cdot \Sigma^*$.
When $\mathcal{S}$ finishes generating a TM configuration c followed by a symbol $m \in \{\forall_l, \exists_l, \exists_r\}$, i.e. $\mathcal{S}$ is in a state of the form $(gen, q, n+1, 1)$, then it chooses nondeterministically between two possible options. Choosing the first option, $\mathcal{S}$ goes to state *cont*, pops m from the stack, and performs Step 3 (see below). Choosing the second option, the behaviour of $\mathcal{S}$ depends on whether c is accepting. If c is *not* accepting (i.e., $q \notin F$), then $\mathcal{S}$ guesses a successor of c going to a state of the form $(gen, q', 0, 0)$ for some $q' \in Q$ without changing the stack content. Therefore, Step 1 is newly performed. If, instead, c is accepting (i.e., $q \in F$), then $\mathcal{S}$ goes to state *rem*, pops m from the stack, and performs Step 2 (see below).
2. *Removing a TM configuration* (*operative phase*) — When $\mathcal{S}$ is in state *rem*, it removes deterministically by pop transitions the TM configuration c on the top of the stack (if any). After having removed c, if the symbol on the top of the stack, say B, belongs to $\{\forall_r, \exists_l, \exists_r\}$ (this means intuitively that $\mathcal{S}$ has already generated a "pseudo" accepting computation tree for the TM configuration currently on the top of the stack), then $\mathcal{S}$ pops B from the stack and goes to state *rem* (i.e., Step 2 is newly performed). If instead $B = \forall_l$, then $\mathcal{S}$ goes to a state of the form $(gen, q', 0, 0)$ for some $q' \in Q$ and replaces $\forall_l$ with the symbol $\forall_r$ on the top of the stack. Therefore, Step 1 is newly performed. Finally, if $B = \gamma_0$ (i.e., the stack is empty), then $\mathcal{S}$ goes to state *fin* and terminates its computation.

[5] Where $k \geq 1$ is a constant such that for each input $y \in \Sigma^*$, the space needed by $\mathcal{M}$ on input y is bounded by $k \cdot |y|$.

3. *Checking δ-consistency* (*control phase*) — When $\mathcal{S}$ is in state *cont*, it checks that one of the following holds:
   - the stack contains exactly one TM configuration.
   - The stack content has the form $c' \cdot m \cdot c \cdot \alpha$ where c and c' are TM configurations and $m \in \{\exists_l, \exists_r, \forall_l, \forall_r\}$.

   In the first case, $\mathcal{S}$ signals success by generating (by its finite control) the symbol *good*. In the second case, $\mathcal{S}$ signals success if and only if c' is a TM successor of c in accordance with m, i.e.: $c' = succ_s(c)$ where $s = l$ *iff* $m \in \{\exists_l, \forall_l\}$. In order to understand how this can be done by using a number of states polynomial in n and $|\mathcal{M}|$, let $c = a_1 \dots a_n$. For each $1 \le i \le n$, the value a'_i of the i-th cell of $succ_l(c)$ (resp., $succ_r(c)$) is completely determined by the values a_{i-1}, a_i and a_{i+1} (taking a_{n+1} for $i = n$ and a_0 for $i = 1$ to be the special symbol "$-$"). As in [15], we denote by $next_l(a_{i-1}, a_i, a_{i+1})$ (resp., $next_r(a_{i-1}, a_i, a_{i+1})$) our expectation for a'_i (these functions can be trivially obtained from the transition function of $\mathcal{M}$). Then, in state *cont*, $\mathcal{S}$ chooses nondeterministically between n states, $cont_1, \dots, cont_n$ without changing the stack content. For each $1 \le i \le n$, if $\mathcal{S}$ is in state $cont_i$, then first, it *deterministically* removes $c' \cdot m$ from the stack, keeping track by its finite control of m and the i-th symbol a'_i of c'. Successively, $\mathcal{S}$ *deterministically* removes c from the stack, keeping also track of the symbols a_{i-1}, a_i, and a_{i+1}. Finally, $\mathcal{S}$ checks whether $a'_i = next_s(a_{i-1}, a_i, a_{i+1})$ with $s = l$ *iff* $m \in \{\exists_l, \forall_l\}$. If this condition is satisfied (and only in this case), then $\mathcal{S}$ generates the symbol *good* and terminates the computation.

Formally, $\mathcal{S} = \langle AP, \Gamma, P, \Delta, L\rangle$ is defined as follows:

- $AP = \{op, cont, good, fin\}$ and $\Gamma = \Sigma \cup (Q \times \Sigma) \cup \{\forall_l, \forall_r, \exists_l, \exists_r\}$;
- $P = \{good, fin, rem\} \cup P_G \cup P_\delta$ where $P_G = \{(gen, q, i, flag) \mid q \in Q,\ 0 \le i \le n+1,\ flag \in \{0,1\},\ flag = 0$ if $i = 0$ and $flag = 1$ if $i = n, n+1\}$ is the set of (control) states used in Step 1, and P_δ, which is used in Step 3, is given by

  $$\begin{aligned}&\{cont, cont_1, \dots, cont_n\} \cup \{(cont_i, j, a) \mid 1 \le i, j \le n \text{ and } a \in \Sigma \cup (Q \times \Sigma)\}\\ &\cup\{(cont_i, j, a, m, a_1, a_2, a_3) \mid\ 1 \le i \le n,\ 0 \le j \le n,\ m \in \{\forall_l, \forall_r, \exists_l, \exists_r\},\\ &\qquad a, a_1, a_2, a_3 \in \Sigma \cup (Q \times \Sigma) \cup \{-\},\ \text{and } a \neq -\}\end{aligned}$$

- $\langle (p, B), (p', \beta)\rangle \in \Delta$ iff one of the following holds:
  - *Step 1* (*generation of a TM configuration*). If $p \in P_G$, then:
    - if $p = (gen, q, i, flag)$ and $i < n$, then $\beta = B'B$ with $B' \in \Sigma \cup (\{q\} \times \Sigma)$ and $p' = (gen, q, i+1, flag')$. Moreover, if $flag = 1$, then $B' \in \Sigma$ and $flag' = 1$; otherwise, $flag' = 0$ iff $B' \in \Sigma$.
    - if $p = (gen, q, n, 1)$, then $p' = (gen, q, n+1, 1)$ and $\beta = B'B$ with $B' = \forall_l$ if $q \in Q_\forall$, and $B' \in \{\exists_l, \exists_r\}$ otherwise.
    - if $p = (gen, q, n+1, 1)$, then or (1) $\beta = \varepsilon$ and $p' = cont$, or (2) $q \notin F$, $\beta = B \in \Gamma$, and $p' = (gen, q', 0, 0)$ for some $q' \in Q$, or (3) $q \in F$, $\beta = \varepsilon$, and $p' = rem$.
  - *Step 2* (*Removing a TM configuration*). If $p = rem$, then:
    - if $B \in \Sigma \cup (Q \times \Sigma) \cup \{\forall_r, \exists_l, \exists_r\}$, then $\beta = \varepsilon$ and $p' = rem$;
    - if $B = \forall_l$, then $\beta = \forall_r$ and $p' = (gen, q', 0, 0)$ for some $q' \in Q$;
    - if $B = \gamma_0$, then $\beta = \varepsilon$, and $p' = fin$.
  - *Step 3* (*Checking δ-consistency*). If $p \in P_\delta$, then:
    - if $p = cont$, then $\beta = B$ and $p' = cont_i$ for some $1 \le i \le n$.
    - If $p = cont_i$, then $B \in \Sigma \cup (Q \times \Sigma)$, $\beta = \varepsilon$, and $p' = (cont_i, 1, B)$;
    - if $p = (cont_i, j, a)$ and $j < n$, then $B \in \Sigma \cup (Q \times \Sigma)$, $\beta = \varepsilon$, and $p' = (cont_i, j+1, a')$ where $a' = B$ if $j = i - 1$, and $a' = a$ otherwise;
    - if $p = (cont_i, n, a)$, then *either* $B = \gamma_0$, $\beta = \varepsilon$, and $p' = good$, *or* $B \in \{\exists_l, \forall_l, \exists_r, \forall_r\}$, $\beta = \varepsilon$, and $p' = (cont_i, 0, a, B, -, -, -)$;
    - if $p = (cont_i, j, a, m, a_1, a_2, a_3)$ and $j < n$, then $B \in \Sigma \cup (Q \times \Sigma)$, $\beta = \varepsilon$, and $p' = (cont_i, j+1, a, m, a'_1, a'_2, a'_3)$ where for each $1 \le h \le 3$, $a'_h = B$ if $j = i + h - 3$, and $a'_h = a_h$ otherwise;
    - if $p = (cont_i, n, a, m, a_1, a_2, a_3)$, then $a = next_s(a_1, a_2, a_3)$ where $s = l$ if and only if $m \in \{\exists_l, \forall_l\}$. Moreover, $\beta = \varepsilon$ and $p' = good$.
- For all $B \in \Gamma \cup \{\gamma_0\}$, $L(good, B) = \{good\}$, $L(fin, B) = \{fin\}$, $L(rem, B) = op$, for all $p \in P_G$, $L(p, B) = \{op\}$, and for all $p \in P_\delta$, $L(p, B) = \{cont\}$.

Let $G_{\mathcal{S}} = \langle W, \to, \mu\rangle$. By construction, the following holds:

Claim Given a TM configuration c with TM state q, there is an accepting computation tree of $\mathcal{M}$ over c *iff* there is a path of $G_{\mathcal{S}}$ of the form $\pi = w_0, w_1, \ldots w_n$ such that $w_0 = ((gen, q, n, 1), c \cdot \gamma_0)$, $\mu(w_n) = fin$, and for each $0 \leq i \leq n-1$, $\mu(w_i) = op$ and if w_i has a successor w'_i such that $\mu(w'_i) = cont$, then each path from w'_i visits a state of the form $(good, \beta)$.

The condition in the claim above can be encoded by the following *CTL* formula

$$\varphi := E\Big(\big[op \wedge AX(cont \rightarrow AF good)\big]\, \mathcal{U}\, fin\Big). \tag{1}$$

Let c_0 be the initial TM configuration (associated with the input x). Then, by the claim above, it follows that $\mathcal{M}$ accepts x *iff* $(G_{\mathcal{S}}, w) \models \varphi$ where $w = ((gen, q_0, n, 1), c_0 \cdot \gamma_0)$. Since φ is independent from $\mathcal{M}$ and n, and the sizes of $|\mathcal{S}|$ and w are polynomial in n and $|\mathcal{M}|$, the theorem holds. □

Theorem 3. *Pushdown model checking against CTL* is* 2EXPTIME*-hard.*

Proof. Let $\mathcal{M} = \langle \Sigma, Q, Q_\forall, Q_\exists, q_0, \delta, F\rangle$ be an EXPSPACE-bounded alternating Turing Machine, and let k be a constant such that for each $x \in \Sigma^*$, the space needed by $\mathcal{M}$ on input x is bounded by $2^{k\cdot|x|}$. Given an input $x \in \Sigma^*$, we define an *PDS* $\mathcal{S}$, a configuration $w_0 = (p_0, \gamma_0)$ of $\mathcal{S}$, and a *CTL** formula φ, whose sizes are *polynomial* in $n = k \cdot |x|$ and in $|\mathcal{M}|$, such that $\mathcal{M}$ accepts x *iff* $(G_{\mathcal{S}}, w_0) \models \varphi$. Some ideas in the proposed reduction are taken from [15], where lower bounds for the satisfiability of extensions of *CTL* and *CTL** are given.

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

Each cell of a TM configuration is coded using a block of n symbols of the stack alphabet of $\mathcal{S}$. The whole block is used to encode both the content of the cell and the location (the number of cell) on the TM tape (note that the number of cell is in the range $[0, 2^n - 1]$ and can be encoded using n bits). The stack alphabet is given by $\{\forall_l, \forall_r, \exists_l, \exists_r\} \bigcup (\Sigma \cup (Q \times \Sigma)) \times 2^{\{b,e,f,cn,l\}}$, where b is used to mark the first element of a TM block, e (resp., f) to mark the first (resp., the last) block of a TM configuration, cn to encode the number of cell, and l to mark a left TM successor.

The behaviour of $\mathcal{S}$ is similar to that of the pushdown system defined in the proof of Theorem 2. The main differences can be summarized as follows:

- *Generation of a TM configuration* (**Step 1**). When $\mathcal{S}$ generates nondeterministically a TM configuration c on the stack, it ensures that each block of c has length n and the symbols b, f, and e are used properly. Moreover, if c is generated as a successor of an other TM configuration, i.e. the stack content before generating c has the form $m \cdot \alpha$ with $m \in \{\exists_l, \exists_r, \forall_l, \forall_r\}$, then $\mathcal{S}$ ensures that the label l is used properly, i.e. any element of c is marked by l iff $m \in \{\exists_l, \forall_l\}$. However, $\mathcal{S}$ does *not* ensure that the cell numbers of c are encoded properly (indeed, this would require a number of control states exponential in n).
- *Generation of the initial TM configuration* — Starting from the global state $w_0 = (p_0, \gamma_0)$, $\mathcal{S}$, first, generates the encoding of the initial TM configuration c_0 (associated with the input x) on the stack. Note that $\mathcal{S}$ ensures that c_0 has the form $(q_0, \sigma_1)\sigma_2 \ldots \sigma_r \#\# \ldots$. However, $\mathcal{S}$ does not ensure that the number of blanks to the right of σ_r is exactly $2^n - r$.
- *Checking δ-consistency* — As for the pushdown system defined in the proof of Theorem 2, after having generated a TM configuration on the stack, $\mathcal{S}$ can choose nondeterministically to go to the (control) state *cont*. When $\mathcal{S}$ is in state *cont*, it chooses nondeterministically between two options $cont_1$ and $cont_2$ (without changing the stack content). Assume that the stack content has the form $c \cdot \alpha$, where c is a “pseudo” TM configuration generated in Step 1, and either α is empty or it has the form $m \cdot c' \cdot \alpha'$ where $m \in \{\exists_l, \exists_r, \forall_l, \forall_r\}$ and c' is a “pseudo” TM configuration. Then, choosing option $cont_1$, $\mathcal{S}$ removes deterministically (by pop transitions) c from the stack and terminates its computation. The computation tree $\langle T, V\rangle$ of $G_{\mathcal{S}}$ rooted at the global state associated with $cont_1$ reduces to a finite path π (corresponding to the configuration c). We use a *CTL** formula φ_1 on this tree $\langle T, V\rangle$ in order to require that the cell numbers of c are encoded correctly (this also implies that the number of blocks of c is exactly 2^n). For each node $u \in \pi$, let $cn(u)$ be the truth value (1 for *true* and 0 for *false*) of the proposition cn in u. Let us consider two consecutive TM blocks $u_1 \ldots u_n u'_1 \ldots u'_n$ along π, and let k (resp., k') be the number of cell of the first block (resp., the second block), i.e., the integer whose binary code is given by $cn(u_1) \ldots cn(u_n)$

(resp., $cn(u'_1)\dots cn(u'_n)$). We have to require that $k' = (k+1)\ mod\ 2^n$, and $k = 0$ (resp., $k' = 2^n - 1$) if $u_1 \dots u_n$ corresponds to the first block of c, i.e. u_1 is labelled by proposition e (resp., $u'_1 \dots u'_n$ corresponds to the last block of c, i.e. u'_1 is labelled by proposition f). Therefore, φ_1 is defined as follows:

$$AG\ \Bigg(\Bigg((b\ \wedge\ e) \rightarrow \bigwedge_{j=0}^{n-1}(AX)^j\ \neg cn\Bigg) \bigwedge \Bigg((b\ \wedge\ f) \rightarrow \bigwedge_{j=0}^{n-1}(AX)^j\ cn\Bigg) \bigwedge$$
$$\Bigg[(b\ \wedge\ \neg f) \longrightarrow \bigvee_{j=0}^{n-1}\Bigg[(AX)^j(\neg cn\ \wedge (AX)^n cn) \wedge$$
$$\bigwedge_{i>j}(AX)^i(cn\ \wedge\ (AX)^n \neg cn)\ \wedge\ \bigwedge_{i<j}(AX)^i(cn\ \leftrightarrow\ (AX)^n cn)\Bigg]\Bigg]\Bigg).$$

Choosing the second option $cont_2$, $\mathcal{S}$, first, removes deterministically c from the stack by pop transitions with the additional ability to generate by its finite control the symbol $check_1$ (this means that the labels of the corresponding configurations of $\mathcal{S}$ contain the proposition $check_1$). Successively, assuming that α has the form $m \cdot c' \cdot \alpha'$, $\mathcal{S}$ removes $m \cdot c'$ from the stack (by pop transitions) and simultaneously generates (by its finite control) at most at one block of c' the symbol $check_2$. After this operation, $\mathcal{S}$ terminates its computation. Let $\langle T, V\rangle$ be the computation tree of $G_{\mathcal{S}}$ rooted at the global state associated with $cont_2$. If α is empty, then by construction, T reduces to a finite path labelled by proposition $check_1$ and corresponding to configuration c. If instead α has the form $m \cdot c' \cdot \alpha'$, then each path (from the root) of T consists of a sequence of nodes corresponding to c labelled by $check_1$, followed by a sequence of nodes corresponding to c' with at most one block labelled by $check_2$. This allows us to define a *CTL** formula φ_2, asserted on the tree $\langle T, V\rangle$, (whose size is polynomial in n and $|\mathcal{M}|$) in order to require that in the case α is not empty (i.e., α has the form $m \cdot c' \cdot \alpha'$), c is a TM successor of c' in accordance with m, i.e. $c = succ_s(c')$ where $s = l$ iff $m \in \{\exists_l, \forall_l\}$ (note that by Step 1, $m \in \{\exists_l, \forall_l\}$ iff c is marked by symbol l). Formula φ_2 is defined as follows:

$$\varphi_2 = AG(\neg check_2)\ \vee\ AG((check_1\ \wedge\ b) \rightarrow E(\theta_1\ \wedge\ \theta_2))$$

where the path formulas θ_1 and θ_2 are defined below. Note that the subformula $AG(\neg check_2)$ manages the case in which α is empty. In the other case, we require that for each node $u \in T$ labelled by $check_1$ and b, i.e. associated with the first element of a block bl of c, there is a path π from u satisfying the following two properties:

1. π visits a node labelled by $check_2$ and b, i.e. associated with the first element of a block bl' of c', such that bl and bl' have the same number of cell. This requirement is specified by the path formula θ_1:

   $$\theta_1 = \psi_1\ \wedge\ X(\psi_2\ \wedge\ X(\psi_3\ \wedge \dots X(\psi_n)\dots))$$

   where for each $1 \le j \le n$, ψ_j is defined as follows

   $$\big(cn\ \rightarrow\ F(check_2 \wedge\ b\ \wedge X^{j-1}\ cn)\big)\ \wedge \big(\neg cn\ \rightarrow\ F(check_2 \wedge\ b\ \wedge X^{j-1} \neg\ cn)\big).$$

2. Let $\widetilde{\Sigma} := \Sigma \cup (Q \times \Sigma)$, and let us denote by $\sigma(\widehat{bl})$ the $\widetilde{\Sigma}$-value of a TM block $\widehat{bl}$. By construction and Property 1 above, there is exactly one node of π that is labelled by $check_2$ and b. Moreover, by Property 1 this node is associated with a TM block bl' of c' having the same number of cell as bl. Therefore, we have to require that $\sigma(bl) = next_s(\sigma(bl_{prec}), \sigma(bl'), \sigma(bl_{succ}))$ where bl_{prec} and bl_{succ} represent the blocks that precede and follow bl' along π, respectively, and $s = l$ iff the TM configuration c is a left TM successor (i.e. all nodes of bl are labelled by proposition l). This requirement is expressed by the path formula θ_2. We distinguish three cases depending on whether bl corresponds to the first block, to the last block, or to a non-extremal block of the associated TM configuration c. For simplicity, we consider only the case in which bl is a non-extremal block. The other cases can be handled similarly.

   $$\theta_2 = (\neg f\ \wedge\ \neg e) \longrightarrow \bigvee\nolimits_{\sigma_1,\sigma_2,\sigma_3 \in \widetilde{\Sigma}}\big(F(\sigma_1\ \wedge (X)^n(\sigma_2\ \wedge b\ \wedge\ check_2\ \wedge\ (X)^n\sigma_3)) \bigwedge$$
   $$(l\ \rightarrow next_l(\sigma_1, \sigma_2, \sigma_3)) \bigwedge (\neg l\ \rightarrow next_r(\sigma_1, \sigma_2, \sigma_3))\big).$$

Finally, formula φ is obtained from formula (1) in the proof of Theorem 2 by replacing the subformula $AX(cont \rightarrow AFgood)$ in (1) with the formula $AX\big[cont \rightarrow \big(EX(cont_1 \wedge \varphi_1) \wedge EX(cont_2 \wedge \varphi_2)\big)\big]$. □

Pushdown systems	**general**	**program complexity**
EF	PSPACE-complete [1,25]	PSPACE-complete [1,25]
CTL	EXPTIME-complete [25]	EXPTIME-complete
*CTL**	2EXPTIME-complete	EXPTIME-complete
Alternation-free modal μ-calculus	EXPTIME-complete [24]	EXPTIME-complete [24]
Modal μ-calculus	EXPTIME-complete [24]	EXPTIME-complete [24]
LTL	EXPTIME-complete [1]	$\in \mathcal{P}$ [1]
Linear-time μ-calculus	EXPTIME-complete [1]	$\in \mathcal{P}$ [1]

Fig. 1. Complexity results on pushdown model checking.

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

Theorem 4. (1) *The program complexity of the PMC problem for CTL is* EXPTIME*-complete.*
(2) *The PMC problem for CTL* is* 2EXPTIME*-complete. The program complexity of the problem is* EXPTIME*-complete.*

Proof. Claim 1 follows from Theorem 2 and the fact that model-checking pushdown systems against *CTL* is known to be EXPTIME-complete [25], while Claim 2 directly follows from Theorems 1 and 3, and Claim 1. □

6. Conclusion

In this paper, we have characterized the complexity of the pushdown model checking problem against *CTL** (which subsumes both *CTL* and *LTL*), showing that it is 2EXPTIME-complete. Moreover, we have shown that the program complexity of the pushdown model checking problem against *CTL* is EXPTIME-complete. Fig. 1 summarizes the main complexity results known in literature about the pushdown model checking problem with respect to standard regular propositional temporal logics. Fig. 1 includes also our complexity results.

Our results confirm that with pushdown systems, the model checking problem is much harder for branching-time temporal logics than for linear-time temporal logics.

An interesting open problem is the complexity (in particular, the program complexity) for both the existential and universal fragments of *CTL**, which subsume *LTL*.

References

[1] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: Application to model-checking, in: Proc. 8th International Conference on Concurrency Theory, CONCUR'97, in: LNCS, vol. 1243, Springer-Verlag, 1997, pp. 135–150.

[2] O. Burkart, B. Steffen, Composition, decomposition and model checking of pushdown processes, Nordic Journal of Computing 2 (2) (1995) 89–125.

[3] D. Caucal, On infinite transition graphs having a decidable monadic theory, in: Proc. 23th International Colloquium on Automata, Languages and Programming, ICALP'96, in: LNCS, vol. 1099, Springer-Verlag, 1996, pp. 194–205.

[4] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer, Alternation, Journal of the ACM 28 (1) (1981) 114–133.

[5] E.M. Clarke, E.A. Emerson, Design and verification of synchronization skeletons using branching time temporal logic, in: Proceedings of Workshop on Logic of Programs, in: LNCS, vol. 131, Springer-Verlag, 1981, pp. 52–71.

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

[7] E.A. Emerson, E.M. Clarke, Using branching time temporal logic to synthesize synchronization skeletons, Science of Computer Programming 2 (3) (1982) 241–266.

[8] E.A. Emerson, J.Y. Halpern, Sometimes and not never revisited: On branching versus linear time, Journal of the ACM 33 (1) (1986) 151–178.

[9] E.A. Emerson, C.S. Jutla, The complexity of tree automata and logics of programs, in: 29th Annual IEEE Symposium on Foundations of Computer Science, FOCS'88, 1988, pp. 328–337.

[10] E.A. Emerson, C.S. Jutla, Tree automata, μ-calculus and determinacy, in: 32nd Annual IEEE Symposium on the Foundations of Computer Science, FOCS'91, 1991, pp. 368–377.

[11] J. Esparza, D. Hansel, P. Rossmanith, S. Schwoon, Efficient algorithms for model checking pushdown systems, in: Proc. 12th International Conference on Computer Aided Verification, CAV'00, in: LNCS, vol. 1855, Springer-Verlag, 2000, pp. 232–247.

[12] J. Esparza, A. Kucera, S. Schwoon, Model checking LTL with regular valuations for pushdown systems, Information and Computation 186 (2) (2003) 355–376.

[13] A. Finkel, B. Willems, P. Wolper, A direct symbolic approach to model checking pushdown systems, Electronic Notes in Theoretical Computer Science 9 (1997).

[14] O. Kupferman, N. Piterman, M.Y. Vardi, Pushdown specifications, in: 9th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'02, in: LNAI, vol. 2514, Springer-Verlag, 2002, pp. 262–277.

[15] O. Kupferman, P.S. Thiagarajan, P. Madhusudan, M.Y. Vardi, Open systems in reactive environments: Control and synthesis, in: Proc. 11th International Conference on Concurrency Theory, CONCUR'00, in: LNCS, vol. 1877, Springer-Verlag, 2000, pp. 92–107.

[16] O. Kupferman, M.Y. Vardi, P. Wolper, An Automata-Theoretic Approach to Branching-Time Model Checking, Journal of the ACM 47 (2) (2000) 312–360.

[17] D.E. Muller, P.E. Shupp, The theory of ends, pushdown automata, and second-order logic, Theoretical Computer Science 37 (1985) 51–75.

[18] D.E. Muller, P.E. Shupp, Alternating automata on infinite trees, Theoretical Computer Science 54 (1987) 267–276.

[19] N. Piterman, M.Y. Vardi, Global model-checking of infinite-state systems, in: Proc. 16th International Conference on Computer Aided Verification, CAV'04, in: LNCS, vol. 3114, Springer-Verlag, 2004, pp. 387–400.

[20] A.P. Sistla, E.M. Clarke, The complexity of propositional linear temporal logics, Journal of the ACM 32 (3) (1985) 733–749.

[21] M.Y. Vardi, A temporal fixpoint calculus, in: 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'88, ACM Press, 1988, pp. 250–259.

[22] M.Y. Vardi, Reasoning about the past with two-way automata, in: Proc. 25th International Colloquium on Automata, Languages and Programming, ICALP'98, in: LNCS, vol. 1443, Springer-Verlag, 1998, pp. 628–641.

[23] M.Y. Vardi, L.J. Stockmeyer, Improved upper and lower bounds for modal logics of programs, in: 17th Annual ACM Symposium on Theory of Computing, STOC'85, 1985, pp. 240–251.

[24] I. Walukiewicz, Pushdown processes: Games and Model Checking, in: Proc. 8th International Conference on Computer Aided Verification, CAV'96, in: LNCS, vol. 1102, Springer-Verlag, 1996, pp. 62–74.

[25] I. Walukiewicz, Model checking CTL properties of pushdown systems, in: Proc. 20th Conference on Foundations of Software Technology and Theoretical Computer Science, FST&TCS'00, in: LNCS, vol. 1974, Springer-Verlag, 2000, pp. 127–138.