Pushdown Module Checking*

Laura Bozzelli† Aniello Murano‡ Adriano Peron§

June 12, 2006

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* (resp., CTL^*) formula, the problem is EXPTIME-complete.

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* ([5, 17, 18]). In this verification method, the behaviour of a system, formally described by a mathematical model, is checked against a behavioural 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 modelled by labelled state-transition graphs.

*A preliminary version of this paper appears in the Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'05, LNCS 3835, Springer-Verlag, 2005, pages 504-518.

†Dipartimento di Matematica e Applicazioni, Università di Napoli "Federico II", Via Cintia, 80126 Napoli, Italy. *E-mail address*: laura.bozzelli@dma.unina.it.

‡Dipartimento di Scienze Fisiche, Università di Napoli "Federico II", Via Cintia, 80126 Napoli, Italy. *E-mail address*: murano@na.infn.it.

§Dipartimento di Scienze Fisiche, Università di Napoli "Federico II", Via Cintia, 80126 Napoli, Italy. *E-mail address*: peron@na.infn.it.

In system modelling, 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 behaviour 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 [9]. 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.

In [14], Kupferman, Vardi, and Wolper 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 labelled 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* [5, 17]. 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 behaviours 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 for 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, it is proved in [14], that the problem is EXPTIME–complete for specifications in *CTL* and 2EXPTIME–complete for specifications in CTL^*. 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 theory of *context-free graphs* is decidable [16]. Concerning *pushdown systems*, Walukiewicz [20] has shown that model checking these 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* [21], which corresponds to a fragment of the alternation-free modal μ-calculus. Recently, in [2], 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 [2]. The situation is quite different for linear-time logics. Model-checking with *LTL* and the *linear-time μ-calculus* is EXPTIME-complete [1]. However, the problem is polynomial 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 [20]. More recently, in [15], 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 [9]). 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 modelled 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 modelled 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^*. 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 pushdown module checking 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 [11]. 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 Section 2, we recall the module checking problem as defined in [14] for both CTL and CTL^* and define open pushdown systems. In Section 3, we recall the framework of *nondeterministic (finite-state) tree automata* and *nondeterministic pushdown tree automata*, which are exploited in Section 4 to solve the pushdown module checking problem against CTL and CTL^*. In Section 4, we describe algorithms to solve the above mentioned problems and give lower bounds that match the upper bounds of the proposed algorithms. We conclude in Section 5.

2 Preliminaries

2.1 Module checking for Branching Temporal Logics

In this subsection we define the module checking problem for CTL and CTL^* [14]. 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 $\{z \in \pi \mid y \preceq z\}$. 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.

The logic CTL^* is a branching–time temporal logic [6], 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 labelled 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 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 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}-labelled trees $\langle T, V\rangle$. Let $x \in T$ and $\pi \subseteq T$ be a path from x. For a state (resp., path) formula φ (resp. θ), the satisfaction relation $(\langle T, V\rangle, x) \models \varphi$ (resp., $(\langle T, V\rangle, \pi) \models \theta$), meaning that φ (resp., θ) holds at node x (resp., holds along path π) in $\langle T, V\rangle$, is defined by induction. The clauses for proposition letters, negation, and conjunction are standard. For the other constructs we have:

- $(\langle T, V\rangle, x) \models A\ \theta$ *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* [14]) $M = \langle AP, W_s, W_e, \rightarrow, w_0, \mu\rangle$, 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-labelled 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* [5, 17]. 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

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

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 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.

Intuitively, each tree in $exec(M)$ corresponds to a different behavior of the environment. In the following, we consider the trees in $exec(M)$ as 2^{AP}-labelled 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* [14]. 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) in a set of *environment* configurations and a set of *system* configurations.

An *OPD* is a tuple $\mathcal{S} = \langle AP, \Gamma, P, p_0, \alpha_0, \Delta, L, Env \rangle$, 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, $\alpha_0 \in \Gamma^* \cdot \gamma_0$ is an initial stack content (where $\gamma_0 \notin \Gamma$ is the *stack bottom symbol*), $\Delta \subseteq (P \times (\Gamma \cup \{\gamma_0\})) \times (P \times \Gamma^*)$ is a finite set of transition rules, $L : P \times (\Gamma \cup \{\gamma_0\}) \to 2^{AP}$ is a labelling 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 $|P| + |\Gamma| + |\alpha_0| + |\Delta|$, with $|\Delta| = \sum_{\langle (p,A),(q,\beta) \rangle \in \Delta} |\beta|$.

An *OPD* $\mathcal{S}$ induces a module $M_{\mathcal{S}} = \langle AP, W_s, W_e, \rightarrow, w_0, \mu \rangle$, 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 $(p, A) \in Env$;
- $w_0 = (p_0, \alpha_0)$;

- $(p, A \cdot \alpha) \rightarrow (q, \beta)$ *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 $\mathcal{M}_S \models_r \psi$.

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) [3] and *Nondeterministic Pushdown Tree Automata* (*PD-NTA*, for short) [11].

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, $\delta : Q \times \Sigma \rightarrow 2^{Q^k}$ is a transition function, and F is an acceptance condition. We consider here *Büchi* and *parity* acceptance conditions [3, 8]. In the case of a parity condition, $F = \{F_1, \ldots, F_m\}$ is a finite sequence of subsets of Q, where $F_1 \subseteq F_2 \subseteq \ldots \subseteq F_m = Q$ (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-labelled k-ary tree $\langle T, V \rangle$ (where $T = \{1, \ldots, k\}^*$) is a Q-labelled 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-labelled (complete) k-ary trees accepted by $\mathcal{A}$.

The *size*$|\mathcal{A}|$ of an *NTA* $\mathcal{A}$ is $|Q| + |\delta| + |F|$ with $|\delta| = \sum_{(q,\sigma) \in Q \times \Sigma} |\delta(q, \sigma)|$.

It is well-known that formulas of *CTL* and *CTL** can be translated to tree automata (accepting the models of the given formula). In particular, we are interested in optimal translations to parity *NTA*. Concerning a *CTL* (resp., *CTL**) formula ψ, given $k \geq 1$, first we build according to [13] a Büchi (resp., parity[2]) *alternating* tree automata $\mathcal{A}$ with $O(|\psi|)$ (resp., $O(2^{|\psi|})$) states and size $O(k \cdot |\psi|)$ (resp., size $O(k \cdot 2^{|\psi|})$ and index $O(|\psi|)$) accepting exactly the complete k-ary trees satisfying ψ. Then, according to [19], we can translate $\mathcal{A}$ into an equivalent parity *NTA* whose size is $O(k \cdot 2^{O(|\psi| \log |\psi|)})$ (resp., $O(k \cdot 2^{2^{O(|\psi|)}})$) and whose index is $O(|\psi|)$ (resp., $O(2^{|\psi|})$).

[2] [13] gives a translation from *CTL** to *Hesitant alternating tree automata* which are a special case of parity alternating tree automata.

Lemma 1 ([13, 19]). *Given a* CTL *(resp., CTL*) formula ψ over AP and $k \geq 1$, we can construct a parity* NTA *of size $O(k \cdot 2^{O(|\psi| \log |\psi|)})$ (resp., $O(k \cdot 2^{2^{O(|\psi|)}})$) and index $O(|\psi|)$ (resp., $O(2^{|\psi|})$) that accepts exactly the set of 2^{AP}-labelled complete k-ary trees that satisfy ψ.*

Remark 1. *Vardi in [19] gives a translation from* (two-way) *alternating parity tree automata $\mathcal{A}$ to parity* NTA *$\mathcal{A}'$. Note that the size of the parity* NTA *$\mathcal{A}'$ is exponential in k. This depends on the fact that Vardi considers arbitrary memoryless strategies of the form $\tau : \{1, \ldots, k\}^* \to 2^{Q \times \{1,\ldots,k\} \times Q}$ where Q is the set of states of $\mathcal{A}$. On the other hand, if $\mathcal{A}$ corresponds to a* CTL *or* CTL* *formula, then any formula of $\mathcal{B}^+(\{1, \ldots, k\} \times Q)$ occurring in the transition function of $\mathcal{A}$ (see [13, 19] for the definition of the transition function of an alternating tree automata) is a positive boolean combination of sub-formulas either of the form $\bigwedge_{i=1}^{i=k}(i, q)$ or of the form $\bigvee_{i=1}^{i=k}(i, q)$ for some $q \in Q$. This means that we can limit ourselves to consider strategies τ such that the following holds for each $x \in \{1, \ldots, k\}^*$ and $(q, i, p) \in \tau(x)$: either $(q, j, p) \notin \tau(x)$ for each $j \neq i$ or $(q, j, p) \in \tau(x)$ for each $1 \leq j \leq k$. This simple observation applied to the algorithm in [19] provides the desired complexity linear in k. This is important, since, as we will see in the next section, k depends on the size of the given pushdown system. Moreover, note that classical translations [18, 7] from* CTL *and* CTL* *to* NTA *lead to* NTA *whose sizes are exponential in k.*

Nondeterministic Pushdown Tree Automata (*PD-NTA*). Here we describe *PD-NTA* (without ε-transitions) over complete k-ary labelled trees. Formally, an *PD-NTA* is a tuple $\mathcal{P} = \langle \Sigma, \Gamma, P, p_0, \alpha_0, \rho, F \rangle$, 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, $\alpha_0 \in \Gamma^* \cdot \gamma_0$ is an initial stack content, $\rho : P \times \Sigma \times (\Gamma \cup \{\gamma_0\}) \to 2^{(P \times \Gamma^*)^k}$ is a transition function, and F is an acceptance condition over P. Intuitively, when the automaton is in state p, reading an input node x labelled by $\sigma \in \Sigma$, and the stack contains a word $A \cdot \alpha$ in $\Gamma^* . \gamma_0$, then the automaton chooses a tuple $\langle (p_1, \beta_1), \ldots, (p_k, \beta_k) \rangle \in \rho(p, \sigma, A)$ and splits in k copies such that for each $1 \leq i \leq k$, 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.

A run of the *PD-NTA* $\mathcal{P}$ on a Σ-labelled k-ary tree $\langle T, V \rangle$ (with $T = \{1, \ldots, k\}^*$) is a $(P \times \Gamma^* . \gamma_0)$-labelled tree $\langle T, r \rangle$ such that $r(\varepsilon) = (p_0, \alpha_0)$ and for each $x \in T$ with $r(x) = (p, A \cdot \alpha)$, there is $\langle (p_1, \beta_1), \ldots, (p_k, \beta_k) \rangle \in \rho(p, V(x), A)$ such that for all $1 \leq i \leq k$, $r(x \cdot i) = (p_i, \beta_i \cdot \alpha)$ if $A \neq \gamma_0$, 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 $inf_r(\pi)$ defined as follows: $inf_r(\pi) \subseteq P$ is the set such that $p \in inf_r(\pi)$ iff there are infinitely many $x \in \pi$ 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 of $\mathcal{P}$, denoted $\mathcal{L}(\mathcal{P})$, contains all trees accepted by $\mathcal{P}$. The *emptiness* problem for *PD-NTA* is to decide, for a given *PD-NTA* $\mathcal{P}$, whether $\mathcal{L}(\mathcal{P}) = \emptyset$.

Proposition 1 ([11]). *The emptiness problem for a parity* PD-NTA *of index m with n states, and transition function ρ can be solved in time exponential in $n \cdot m \cdot |\rho|$ with $|\rho| = \sum_{\langle (p_1,\beta_1),\ldots,(p_k,\beta_k)\rangle \in \rho(p,\sigma,A)} |\beta_1| + \ldots + |\beta_k|$.*

PD-NTA are closed under intersection with *NTA*.

Proposition 2. *For a Büchi* PD-NTA $\mathcal{P} = \langle \Sigma, \Gamma, P, p_0, \alpha_0, \rho, F\rangle$ *with $F = P$ and a parity* NTA $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F'\rangle$*, there is 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 $\mathcal{A}$, and the size of the transition relation is bounded by $|\rho| \cdot |\delta|$.*

Proof. The *PD-NTA* $\mathcal{P}'$ is defined as $\mathcal{P}' = \langle \Sigma, \Gamma, Q \times P, (q_0, p_0), \alpha_0, \rho', F''\rangle$ such that $\langle((q_1,p_1),\beta_1),\ldots,((q_k,p_k),\beta_k)\rangle \in \rho'((q,p),\sigma,A)$ iff $\langle(p_1,\beta_1),\ldots,(p_k,\beta_k)\rangle \in \rho(p,\sigma,A)$ and $\langle q_1,\ldots,q_k\rangle \in \delta(q,\sigma)$. Moreover, if $F' = \{F_1,\ldots,F_m\}$, then $F'' = \{F_1 \times P, \ldots, F_m \times P\}$. □

4 Deciding Pushdown Module Checking

In this section we solve the pushdown module checking for both *CTL* and *CTL**. First, in subsection 4.1 we give an algorithm based on an automata-theoretic approach. Then, in subsection 4.2 we give lower bounds that match the upper bounds provided by our algorithm.

4.1 Upper Bounds

We fix an *OPD* $\mathcal{S} = \langle AP, \Gamma, P, p_0, \alpha_0, \Delta, L, Env\rangle$ and a *CTL* (resp., *CTL**) formula ψ. We solve the pushdown module-checking problem for $\mathcal{S}$ against ψ using an automata-theoretic approach: we construct a parity *PD-NTA* $\mathcal{P}_{\mathcal{S}\times\neg\psi}$ as the intersection of two tree automata. Essentially, the first automaton, denoted by $\mathcal{P}_{\mathcal{S}}$, is a Büchi *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 [14] 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}-labelled 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 $2^{AP\cup\{t\}} \cup \{\bot\}$-labelled 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 labelled by $\bot$ we add recursively k-children labelled 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 $2^{AP\cup\{t\}} \cup \{\bot\}$-labelled 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 labelled 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 labelling. Thus, the proposition $\bot$ is used to denote both "disabled" states and "completion" states. 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 labelled by $\bot$. Let us denote by $\widehat{exec}(M_{\mathcal{S}})$ the set of all $2^{AP\cup\{t\}} \cup \{\bot\}$-labelled k-ary trees obtained from $\langle \{1,\ldots,k\}^*, V'\rangle$ in the above described manner. The Büchi *PD-NTA* $\mathcal{P}_{\mathcal{S}} = \langle \Sigma, \Gamma, P', (p_0, \top), \alpha_0, \rho, P'\rangle$, 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. Let $p \in P$ and $A \in \Gamma \cup \{\gamma_0\}$ with $next_{\mathcal{S}}(p, A) = \langle (p_1, \beta_1), \ldots, (p_d, \beta_d)\rangle$ (where $0 \le d \le k$). Then, for $m \in \{\top, \vdash, \bot\}$ and $\sigma \in \Sigma$, $\rho((p, m), \sigma, A) \neq \emptyset$ *iff* one of the following holds (where $\alpha = A$ if $A \in \Gamma$, and $\alpha = \varepsilon$ otherwise):
  - $\sigma = \bot$ and $m \in \{\vdash, \bot\}$. In this case we have
$$\rho((p, m), \bot, A) = \{\langle \underbrace{((p, \bot), \alpha), \ldots, ((p, \bot), \alpha)}_{k \text{ pairs}} \rangle\}$$
That is, $\rho((p, m), \bot, A)$ contains exactly one k-tuple. In this case all the successors of the current configuration are disabled.
  - $\sigma \neq \bot$, $m \in \{\vdash, \top\}$, and $next_{\mathcal{S}}(p, A)$ is empty (i.e., $d = 0$). In this case $\sigma = L(p, A) \cup \{t\}$ (i.e., the current configuration is terminal) and
$$\rho((p, m), L(p, A) \cup \{t\}, A) = \{\langle((p, \bot), \alpha), \ldots, ((p, \bot), \alpha)\ \rangle\}$$
  - $\sigma \neq \bot$, $(p, A) \notin Env$, $m \in \{\vdash, \top\}$, and $next_{\mathcal{S}}(p, A)$ is *not* empty (i.e., $d \ge 1$). In this case $\sigma = L(p, A)$ and $\rho((p, m), L(p, A), A)$ is given by
$$\{\langle((p_1, \top), \beta_1), \ldots, ((p_d, \top), \beta_d), \underbrace{((p, \bot), \alpha), \ldots, ((p, \bot), \alpha)}_{k-d \text{ pairs}} \rangle\}$$

– $\sigma \neq \bot$, $(p, A) \in Env$, $m \in \{\vdash, \top\}$, and $next_{\mathcal{S}}(p, A)$ is *not* empty (i.e., $d \geq 1$). In this case $\sigma = L(p, A)$ and $\rho((p, m), L(p, A), A)$ is given by

$$\begin{array}{l}\{\ \langle((p_1,\top),\beta_1),\ ((p_2,\vdash),\beta_1),\ldots,((p_d,\vdash),\beta_d),\ ((p,\bot),\alpha),\ldots,((p,\bot),\alpha)\rangle,\\ \ \ \langle((p_1,\vdash),\beta_1),\ ((p_2,\top),\beta_1),\ldots,((p_d,\vdash),\beta_d),\ ((p,\bot),\alpha),\ldots,((p,\bot),\alpha)\rangle,\\ \qquad\qquad\qquad\qquad\qquad\vdots\\ \ \ \langle((p_1,\vdash),\beta_1),\ ((p_2,\vdash),\beta_1),\ldots,((p_d,\top),\beta_d),\ ((p,\bot),\alpha),\ldots,((p,\bot),\alpha)\ \rangle\}.\end{array}$$

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.

Note that $\mathcal{P}_{\mathcal{S}}$ has $3 \cdot |P|$ states, and $|\rho|$ is bounded by $k(|P| \cdot |\Gamma| + |\Delta|)$. Assuming that $|P| \cdot |\Gamma| \leq |\Delta|$, we have that $|\rho| \leq k \cdot |\Delta|$.

We recall that a node labelled by $\bot$ 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 $\langle T, V\rangle$ (that we call "legal" paths) that either never visit a node labelled by $\bot$ or contain a *terminal* node (i.e. a node labelled by t). Note that a path is *not* "legal" iff it satisfies the formula $\neg t\ \mathcal{U}\ \bot$. In order to achieve this, as in [14] 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 [14], 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 $prop \in AP$;
- $f(\neg\varphi) = \neg f(\varphi)$;
- $f(\varphi_1 \wedge \varphi_2) = f(\varphi_1) \wedge f(\varphi_2)$;
- $f(E\theta) = E((G\neg\bot) \wedge f(\theta))\ \vee\ E((F\ t) \wedge f(\theta))$;
- $f(A\theta) = A((\neg t\ \mathcal{U}\ \bot)\ \vee\ f(\theta))$;
- $f(X\theta) = X(f(\theta) \wedge \neg\bot)$;
- $f(\theta_1\ \mathcal{U}\ \theta_2) = (f(\theta_1) \wedge \neg\bot)\ \mathcal{U}\ (f(\theta_2) \wedge \neg\bot)$.

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 [10], 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}-labelled tree obtained from $\langle T, V\rangle$ removing all the nodes labelled by $\bot$ (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 Büchi *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-labelled complete k-ary trees that are models of $f(\neg\psi)$

(recall that $\Sigma = 2^{AP \cup \{t\}} \cup \{\bot\}$). By Lemma 1, if ψ is a *CTL* (resp., *CTL**) formula, then $\mathcal{A}_{\neg\psi}$ has size $O(k \cdot 2^{O(|\psi| \log |\psi|)})$ (resp., $O(k \cdot 2^{2^{O(|\psi|)}})$) and index $O(|\psi|)$ (resp., $O(2^{|\psi|})$). Therefore, by Proposition 2 the following holds:

- If ψ is a *CTL* formula, then $\mathcal{P}_{\mathcal{S} \times \neg\psi}$ has $O(k \cdot |P| \cdot 2^{O(|\psi| \log |\psi|)})$ states, index $O(|\psi|)$, and transition relation bounded by $O(k^2 \cdot |\Delta| \cdot 2^{O(|\psi| \log |\psi|)})$.
- If ψ is a *CTL** formula, then $\mathcal{P}_{\mathcal{S} \times \neg\psi}$ has $O(k \cdot |P| \cdot 2^{2^{O(|\psi|)}})$ states, index $O(2^{|\psi|})$, and transition relation bounded by $O(k^2 \cdot |\Delta| \cdot 2^{2^{O(|\psi|)}})$.

Therefore, by Proposition 1 we obtain the main result of this subsection.

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.

4.2 Lower Bounds

In this section we give lower bounds for the considered problems that match the upper bounds of the algorithm proposed in the previous subsection. 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 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[3]. A node that corresponds to a universal configuration c (i.e. the associated

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

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 c (i.e. the associated state is in $Q_\exists$) has a single successor, corresponding to either $succ_l(c)$ or $succ_r(c)$. The tree is accepting iff all its paths (from the root) reach an accepting configuration. An input $x \in \Sigma^*$ is *accepted* by $\mathcal{M}$ iff there exists an accepting computation tree of $\mathcal{M}$ on x.

If $\mathcal{M}$ is EXPSPACE–bounded (resp., 2EXPSPACE–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 $2^{k \cdot |x|}$ (resp., $2^{2^{k \cdot |x|}}$). It is well-known [4] that 2EXPTIME (resp., 3EXPTIME) coincides with the class of all languages accepted by EXPSPACE–bounded (resp., 2EXPSPACE–bounded) alternating Turing Machines.

In the following we fix an input word x and let $n = k \cdot |x|$.

Theorem 2.

(1) *The pushdown module checking problem for* CTL *is* 2EXPTIME*–hard.*

(2) *The pushdown module checking problem for* CTL* *is* 3EXPTIME*–hard.*

Proof. Here we give the proof for *CTL*. The proof for *CTL** is reported in Appendix. For the EXPSPACE–bounded alternating Turing Machine $\mathcal{M} = \langle \Sigma, Q, Q_\forall, Q_\exists, q_0, \delta, F \rangle$ and the input x, we build an *OPD* $\mathcal{S}$ and a *CTL* formula φ whose sizes are *polynomial* in n and in $|\mathcal{M}|$ such that $\mathcal{M}$ accepts x *iff* there is a tree $\langle T, V \rangle \in exec(M_\mathcal{S})$ such that $\langle T, V \rangle$ satisfies φ, i.e. *iff* $M_\mathcal{S} \not\models_r \neg\varphi$. Some ideas in the proposed reduction are taken from [12], where there are given lower bounds for the satisfiability of extensions of *CTL* and *CTL**.

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 $x = \sigma_1 \dots \sigma_r$ (where $r = |x|$), then the initial configuration is given by $(q_0, \sigma_1)\sigma_2 \dots \sigma_r \underbrace{\#\# \dots \#}_{2^n - r}$.

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 $(\Sigma \cup (Q \times \Sigma)) \times 2^{\{b, fc, e, cn, l\}}$ where b is used to mark the first element of a TM block, fc to mark the initial TM configuration, e to mark the first element of the first block of a TM configuration, cn to encode the number of cell, and l to mark a left TM successor. The pushdown system $\mathcal{S}$ proceeds in two phases.

Phase 1 Starting from the initial configuration (with empty stack content), $\mathcal{S}$ generates nondeterministically by push transitions a sequence of TM configurations on the stack. $\mathcal{S}$ ensures that the first TM configuration is the initial TM configuration (corresponding to the input x). Moreover, the following conditions are satisfied for any generated TM configuration c:

- $\mathcal{S}$ ensures that the symbols b, fc, and e are used properly. Moreover, $\mathcal{S}$ ensures that the last block of c is the unique block in c that has number of cell $2^n - 1$ (i.e, all its elements are marked by symbol cn).

- All global states of $\mathcal{S}$ associated with all elements of c except[4] the last element are *environment states*. $\mathcal{S}$ keeps track of the TM state q associated with c by its finite control. If c is not accepting (i.e., $q \notin F$), then the global state s associated with the last element of c is a *system state* if c is a TM universal configuration (i.e., $q \in Q_\forall$), and it is an *environment state* otherwise. In such a state s, $\mathcal{S}$ without modifying the stack content chooses a letter 0/1 to encode the choice of the transition in $\mathcal{M}$. According to such a choice all elements of the next TM configuration will be marked by the corresponding choice symbol. In particular, we use the symbol l to mark all elements of a TM left successor.

Note that $\mathcal{S}$ does *not* ensure that the number of blocks of any generated TM configuration is exactly 2^n (i.e., the cell numbers are updated correctly) and the generated TM configuration sequence is consistent with the transition function of $\mathcal{M}$. In particular, concerning the initial TM configuration, $\mathcal{S}$ ensures that it has the form $(q_0, \sigma_1)\sigma_2 \dots \sigma_r \# \# \dots$, but $\mathcal{S}$ does not ensure that the number of blanks to the right of σ_r is exactly $2^n - r$.

Phase 2 After having generated an accepting configuration, $\mathcal{S}$ reaches a *system global state* in which chooses between two possible options opt_1 and opt_2 (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 linear path π that corresponds to the sequence ν of "pseudo" TM configurations generated in the first phase in reversed order. We use this subtree together with a *CTL* formula φ_{opt_1} to check that the cell numbers of the sequence ν are encoded correctly (this implies that each configuration of ν has exactly length 2^n). For each node u, 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 blocks $u_n \dots u_1 u'_n \dots u'_1$ along π (note that these two blocks appear in reversed order w.r.t. the corresponding blocks in ν), 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 $cn(u_n) \dots cn(u_1)$ (resp., $cn(u'_n) \dots cn(u'_1)$). We have to require that $k = (k' + 1)\ mod\ 2^n$, and $k = 0$ iff $u_n \dots u_1$ is the first block of a TM configuration (i.e. u_1 is labelled by proposition e). Therefore, formula φ_{opt_1} is defined as follows:

$$
\begin{array}{ll}
AG\ \big(& (AX)^{n-1}(b\ \wedge \neg(e\ \wedge\ fc)) \longrightarrow \\
& \big[\bigvee_{j=0}^{n-1} \big[\ (AX)^j(cn\ \wedge (AX)^n \neg cn)\ \wedge\ \bigwedge_{i>j}(AX)^i(\neg cn\ \wedge\ (AX)^n cn) \wedge \\
& \qquad \bigwedge_{i<j}(AX)^i(cn\ \leftrightarrow\ (AX)^n cn)\big]\ \wedge\ (AX)^{n-1}\neg e\ \big]\ \bigvee \\
& \big[\bigwedge_{j=0}^{n-1}[(AX)^j(\neg cn\ \wedge\ (AX)^n cn)]\ \wedge\ (AX)^{n-1}e\big]\ \big)
\end{array}
$$

By selecting opt_2, $\mathcal{S}$ empties the stack by a sequence of pop transitions with the additional ability to generate exactly at one block (corresponding to a TM cell) the symbol $check_1$ and successively[5] exactly at one block the symbol $check_2$. Therefore, a computation in this phase corresponds to the sequence ν of "pseudo"

[4] for elements of c, we mean the stack symbol occurrences associated with the encoding of c

[5] if the $check_1$-block does not correspond to the first block of ν

TM configurations generated in the first phase in *reversed order* with exactly one block marked by $check_1$ and exactly one block marked by $check_2$. Let $\langle T, V\rangle$ be the corresponding subtree of the computation tree of $M_{\mathcal{S}}$. By construction of $\mathcal{S}$, the following holds:

1. Each node of T has at most two successors. For each path π of T (from the root), the nodes that precede the $check_1$-block correspond to *system states*, while the nodes that follow the $check_1$-block correspond to *environment states*. Moreover, if a node has two successors, then it is labelled by b (i.e., it corresponds to the first element of a TM block). Also, each b-node following the $check_1$-block and preceding the $check_2$-block (if any) has two successors and exactly one of these two successors is labelled by $check_2$.

We use the subtree $\langle T, V\rangle$ together with a *CTL* formula φ_{opt_2} in order to check that ν is faithful to the evolution of $\mathcal{M}$.
In order to understand how this can be done, let $c = a_1 \dots a_{2^n}$ be a TM configuration. For any $1 \leq i \leq 2^n$, 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_{2^n+1} for $i = 2^n$ and a_0 for $i = 1$ to be some special symbol). As in [12], 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 $\mathcal{M}$).

Let $exec(\langle T, V\rangle)$ be the set of the trees obtained by pruning from $\langle T, V\rangle$ subtrees whose root is a child of a node corresponding to an environment state. Note that by Property 1, each tree $\langle T', V'\rangle \in exec(\langle T, V\rangle)$ satisfies the following:

- For each block bl of ν, there is a path in T' (from the root) such that the sequence of nodes associated with bl is labelled by $check_1$.

Formula φ_{opt_2} will capture all trees $\langle T', V'\rangle \in exec(\langle T, V\rangle)$ satisfying the following:

A. For each $u \in T'$ labelled by $check_1$, there is exactly one path in T' from u. By Property 1, we can express this requirement as follows:

$$\varphi_A := AG\ \big(\ \neg(EX\ check_2\ \wedge\ EX\ \neg check_2)\big)$$

B. Let $u \in T'$ be a b-node labelled by $check_1$ and associated with a block bl_1 that does not belong to the first TM configuration. Then, each path π of T' from u (note that by Property **A** above such a path is uniquely determined) visits a b-node u' associated with a block bl_2 marked by $check_2$ such that bl_1 and bl_2 belong to two consecutive TM configurations along ν and have the same cell number. Concerning the first requirement, it suffices to specify that there is exactly a node between u and u' (excluding u') in which proposition e holds (we recall that e marks the first element of the first block of a TM configuration). Therefore, this requirement can be expressed as follows:

$$\varphi_{B.1} := AG\ \big(\quad (b\ \wedge check_1\ \wedge\ \neg fc) \longrightarrow A\ (\neg e)\ \mathcal{U}\ \big(e\ \wedge AX\ A\ ((\neg e)\ \mathcal{U}\ (b\ \wedge check_2))\big)\quad\big)$$

The second requirement can be expressed as follows:

$$\varphi_{B.2} := AG\,\Big(\quad (check_1\ \wedge\ (AX)^{n-1}\,(b\ \wedge check_1\ \wedge \neg fc)) \longrightarrow \\ [\psi_1\ \wedge\ AX(\psi_2\ \wedge\ AX(\psi_3\ \wedge \ldots AX(\psi_n)\ldots))]\quad \Big)$$

where for any $1 \leq j \leq n$, the formula ψ_j is defined as follows

$$\psi_j ::= \begin{array}{l} \big(cn\ \rightarrow\ AF(check_2 \wedge\ cn\ \wedge (AX)^{n-j}(b\ \wedge\ check_2)\big)\ \bigwedge \\ \big(\neg cn\ \rightarrow\ AF(check_2 \wedge\ \neg cn\ \wedge (AX)^{n-j}(b\ \wedge\ check_2)\big) \end{array}$$

C. Let $\widetilde{\Sigma} := \Sigma \cup (Q \times \Sigma)$. Let $u \in T'$ be a b-node labelled by $check_1$ and with $\widetilde{\Sigma}$-label σ (and which does not belong to the first TM configuration), and let u' with $\widetilde{\Sigma}$-label σ' be the b-node labelled by $check_2$ belonging to the unique path π from u (according to property **A**). Finally, let σ_p and σ_s be the $\widetilde{\Sigma}$-labels of the b-nodes following u' and preceding u' along π respectively. By Property **B** u' corresponds to a TM block with the same number of cell as u and belonging to the precedent TM configuration w.r.t. ν. Thus, we have to require that $\sigma = next_l(\sigma_p, \sigma', \sigma_s)$ if u corresponds to a block of the left successor of the configuration associated with u' (i.e., u is labelled by proposition l), and $\sigma = next_r(\sigma_p, \sigma', \sigma_s)$ otherwise. This requirement is expressed by the following CTL formula φ_C. We distinguish three cases depending on whether u belongs to the first block, to the last block or to a non-extremal block of the associated TM configuration. For simplicity, we consider only the case in which u belongs to a non-extremal block. The other cases can be handled similarly.

$$\varphi_C := AG\,\Big(\quad \Big[(b\ \wedge check_1\ \wedge\ \neg fc\ \wedge\ \neg e\ \wedge\ \ \neg AF(e\ \wedge AX\ check_2)) \longrightarrow \\ \big(\bigvee_{\sigma_1,\sigma_2,\sigma_3 \in \widetilde{\Sigma}}\ AF(\sigma_1\ \wedge (AX)^n(\sigma_2\ \wedge b\ \wedge\ check_2\ \wedge\ (AX)^n\sigma_3) \\ \bigwedge (l\ \rightarrow next_l(\sigma_3, \sigma_2, \sigma_1))\ \bigwedge \\ (\neg l\ \rightarrow next_r(\sigma_3, \sigma_2, \sigma_1)))\Big]\quad \Big)$$

Therefore, φ_{opt_2} is given by $\varphi_A\ \wedge\ \varphi_{B.1}\ \wedge \varphi_{B.2}\ \wedge\ \varphi_C$. It is clear that assuming that the cell numbers of ν are encoded correctly (this is guaranteed by formula φ_{opt_1}), then ν is a legal sequence of TM configurations iff there is $\langle T', V' \rangle \in exec(\langle T, V \rangle)$ satisfying φ_{opt_2}.

By considerations above, we deduce that $\mathcal{M}$ accepts x iff there is $\langle T, V \rangle \in exec(M_{\mathcal{S}})$ such that each path of T (from the root) reaches a node u corresponding to the last element of an accepting TM configuration and the following holds: the subtree associated with the opt_1-child (resp., opt_2-child) of u satisfies formula φ_{opt_1} (resp., φ_{opt_2}). Therefore, formula φ is defined as follows:

$$AF\big(EX(opt_1\ \wedge\ \varphi_{opt_1})\ \wedge\ EX(opt_2\ \wedge\ \varphi_{opt_2})\big)$$

The construction of the *OPD* $\mathcal{S}$ is simple. Thus, we omit the details. □

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

Theorem 3.

(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 and 2. 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 [2]. Thus, Claim 3 follows from Theorem 1. □

5 Conclusion

In [14], module checking has been introduced 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 1 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 (except for the system complexity of the pushdown model checking against *LTL*). Our complexities results provide an additional evidence that for pushdown systems, checking *CTL* or *CTL** properties is actually harder than checking *LTL* properties.

	Model Checking	System complexity of Model Checking	Module Checking	System complexity of Module Checking
LTL	EXPTIME [1]	$\in$ PTIME [1]	EXPTIME	$\in$ PTIME
CTL	EXPTIME [21]	EXPTIME [2]	2EXPTIME	EXPTIME
CTL*	2EXPTIME [2]	EXPTIME [2]	3EXPTIME	EXPTIME

Figure 1: Complexity results on pushdown module checking and pushdown model checking

A Appendix

A.1 Lower bound for CTL^*

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 $x \in \Sigma^*$, the space needed by $\mathcal{M}$ on input x is bounded by $2^{2^{k \cdot |x|}}$. Given an input $x \in \Sigma^*$, we build an *OPD* $\mathcal{S}$ 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* there is a tree $\langle T, V \rangle \in exec(M_\mathcal{S})$ such that $\langle T, V \rangle$ satisfies φ, i.e. *iff* $M_\mathcal{S} \not\models_r \neg\varphi$.

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 2^{2^n}. If $x = \sigma_1 \dots \sigma_r$ (where $r = |x|$), then the initial configuration is given by $(q_0, \sigma_1)\sigma_2 \dots \sigma_r \underbrace{\#\# \dots \#}_{2^{2^n} - r}$.

As in [12] we use two counters to encode the number of a TM cell. 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 $n \cdot 2^n$ symbols of the stack alphabet of $\mathcal{S}$. This means that each block is a sequence of 2^n sub-blocks of length n, 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. Moreover, the content of the cell (a symbol in $\Sigma \cup (Q \times \Sigma)$) is maintained in the first element of the block. We use the proposition cn_1 (resp., cn_2) to encode the value of each bit of the 2^n-bit (resp., n-bit) counter. For instance, a sub-block in which each element is marked by cn_2 corresponds to the last sub-block of the given block.

As the *OPD* for the *CTL* case, $\mathcal{S}$ proceeds in two phases. The first phase, in which $\mathcal{S}$ generates nondeterministically a sequence of "pseudo" TM configurations on the stack, is identical to that for the *CTL* case, with the only difference that $\mathcal{S}$, in addition, uses the proposition *sb* to mark the first element of a TM sub-block, and the proposition cn_2 to encode the n-bit counter. Note that in this phase $\mathcal{S}$ does *not* ensure that the number of blocks of any generated TM configuration is exactly 2^{2^n}, the number of sub-blocks of each TM block is exactly 2^n, and the generated configuration sequence is consistent with the transition function of $\mathcal{M}$.

After having generated an accepting configuration, $\mathcal{S}$ reaches a *system global state* in which chooses between two possible options opt_1 and opt_2. Let ν be the sequence of TM configurations generated in the first phase. We use the subtree rooted at opt_1 together with a suitable CTL^* formula φ_{opt_1} to check that the cell numbers of the sequence ν have been encoded correctly (i.e., the 2^n-bit and n-bit counters have been encoded properly). This part of the construction is not very difficult, and therefore, we omit it. Finally, we use the subtree rooted at opt_2, say $\langle T_2, V_2 \rangle$, together with a CTL^* formula φ_{opt_2} in order to check that ν is faithful to the evolution of $\mathcal{M}$. Since this part of the construction is not trivial, we describe it in detail.

By selecting opt_2, $\mathcal{S}$ empties the stack by a sequence of pop transitions and at

the same time, it performs the following operations:

1. $\mathcal{S}$ generates nondeterministically exactly at one TM block bl of ν the symbol $check_1$. All the global states associated with TM blocks that precede bl are system states.

2. After having generated at a TM block the symbol $check_1$, $\mathcal{S}$ eventually and nondeterministically selects an option, opt_3, at the end of a TM block bl (considering ν in reversed order) without changing the stack content. The corresponding pushdown configuration is a system state. In such a state s, $\mathcal{S}$ chooses, without changing the stack content, two possible sub-options: opt_3^1 and opt_3^2. Let bl_1 be the TM block (if any) following bl (considering ν in reversed order), and let bl_2 be the TM block (if any) following bl_1. Then:

   2.1. By choosing opt_3^1, $\mathcal{S}$ empties the stack deterministically and generates at TM block bl_1 (resp., bl_2) the symbol $curBlock$ (resp., $precBlock$).

   2.2. Choosing opt_3^2, $\mathcal{S}$ empties the stack with the additional ability to generate nondeterministically at most at one sub-block of bl_1 the symbol $curSubBlock$. All global states of $\mathcal{S}$ in this phase are system states.

3. All global states that precede opt_3 and follow $check_1$ are environment states.

Figure 2 (resp., 3) shows the structure of a subtree rooted at opt_2 (resp., opt_3).

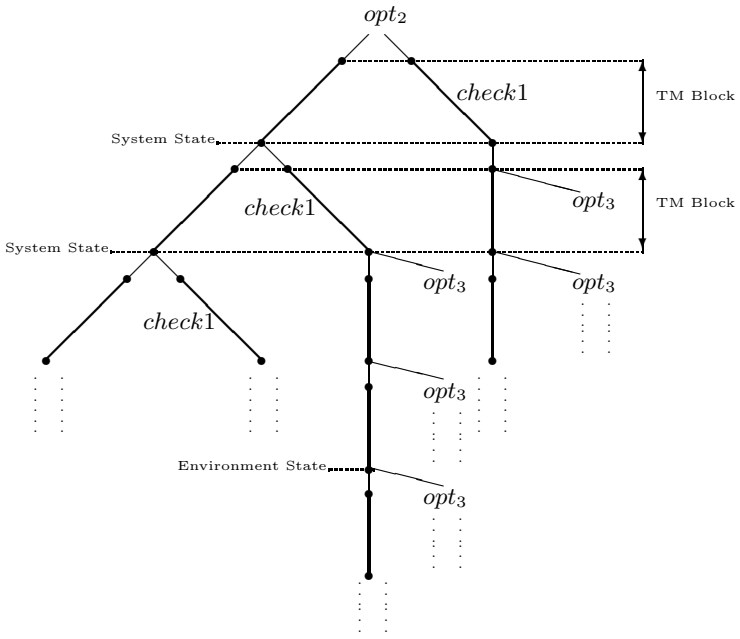

Figure 2: Structure of a subtree rooted at a opt_2 node.

Formula φ_{opt_2} captures all $\langle T, V\rangle \in exec(\langle T_2, V_2\rangle)$ satisfying the following:

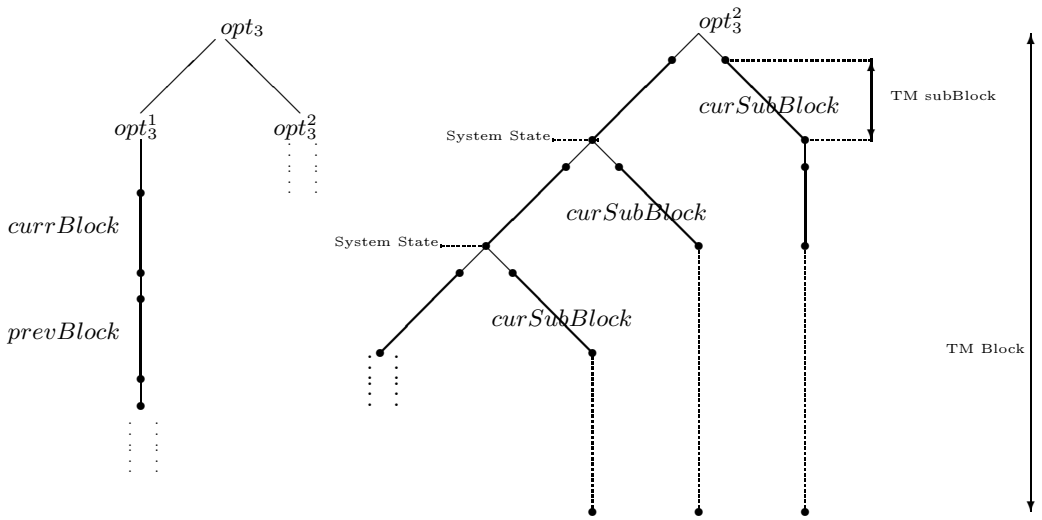

Figure 3: Structure of a subtree rooted at an opt_3 node.

A. For each TM block bl of ν, there is a path π in $\langle T, V\rangle$ such that the sequence of nodes associated with bl is labelled by $check_1$. By Property 1 above, each $\langle T, V\rangle \in exec(\langle T_2, V_2\rangle)$ satisfies this requirement.

B. Let $u \in T$ be a b-node labelled by $check_1$ and associated with a TM block bl, and let T_u be the subtree of T rooted at u. Moreover, let $T_{2,u}$ be the subtree of T_2 rooted at u. By Property 2 above, for each TM block bl' that precedes bl w.r.t. ν, there is a path π of $T_{2,u}$ such that all and only the nodes of π corresponding to block bl' are labelled by proposition $curBlock$. Since we have to check consistency of ν with respect to functions $next_l$ and $next_r$, for the given block bl (assuming that it does not belong to the initial TM configuration) we want to be able to select the TM block having the same number of cell as bl and belonging to the previous TM configuration w.r.t. ν. In order to achieve this, first we have to allow the selection of at most one TM block bl' labelled by $curBlock$ in the tree T_u. By construction (see Figures 2 and 3) this requirement can be specified as

$$\varphi_B := AG\ \big(\ \neg(EX\ opt_3\ \wedge\ EX\ \neg opt_3)\big)$$

C. For each b-node $u \in T$ labelled by $check_1$ (and which does not belong to the first TM configuration of ν), there is a path π from u that visits a b-node u' labelled by $currBlock$ (note that by Property **B** above there is at most one path from u satisfying this requirement). Moreover, the TM blocks bl and bl' associated with u and u', respectively, have the same cell number and belong to successive configurations w.r.t. ν. Concerning the second requirement, this can be expressed as follows:

$$\varphi_{C.1} := AG\ \big(\quad (b\ \wedge check_1\ \wedge\ \neg fc) \longrightarrow \\ E\ \big(\neg e\ \mathcal{U}\ (e\ \wedge X\ (\neg e\ \mathcal{U}\ (b\ \wedge currBlock)))\ \big)\quad \big)$$

Formula $\varphi_{C.1}$ asserts that there is exactly a node between u and u' (excluding u') in which e holds (we recall that proposition e marks the first element of the first block of a TM configuration).
Now, let us consider the first requirement. Formulas φ_B and $\varphi_{C.1}$ ensure that there is exactly one node labelled by opt_3 in the subtree of T rooted at u. Let v be the successor of such a node labelled by opt_3^2 (according to Property 2 above) and let T_v be the subtree of T rooted at v. By Property 2.2 above, for each sub-block of bl' there is exactly one path in T_v such that all and only the nodes corresponding to such sub-block are labelled by the proposition $curSubBlock$. Therefore, it suffices to require that for each sb-node w corresponding to the first element of some sub-block sbl of bl there is a path π from w that visits a sub-block sbl' labelled by $curSubBlock$ such that (i) the value of the n-bit counter at two sub-blocks sbl_1 and sbl_2 is the same (i.e., sbl_1 and sbl_2 have the same position w.r.t. the associated TM block) and (ii) the *bit value* of the 2^n-bit counter at sbl_1 and sbl_2 is the same. This requirement can be expressed by the following CTL^* formula

$$\varphi_{C.2} := AG\ \Big(\quad (check_1\ \wedge\ (EX)^{n-1}\ (sb\ \wedge check_1\ \wedge \neg fc)) \longrightarrow \\ E[\psi_1\ \wedge\ X(\psi_2\ \wedge\ X(\psi_3\ \wedge \ldots \wedge X(\psi_n\ \wedge\ \phi)\ldots))]\quad \Big)$$

where for any $1 \leq j \leq n$, the path formula ψ_j is defined as follows:

$$\big(cn_2\ \rightarrow\ F(curSubBlock \wedge\ cn_2\ \wedge (X)^{n-j}(sb\ \wedge\ curSubBlock)\big)\ \bigwedge \\ \big(\neg cn_2\ \rightarrow\ F(curSubBlock \wedge\ \neg cn_2\ \wedge (X)^{n-j}(sb\ \wedge\ curSubBlock)\big)$$

end the path formula ϕ is defined as follows:

$$cn_1\ \rightarrow\ F(curSubBlock \wedge\ sb\ \wedge cn_1) \bigwedge \\ \neg cn_1\ \rightarrow\ F(curSubBlock \wedge\ sb\ \wedge \neg cn_1)$$

We recall that proposition cn_1 (resp., cn_2) is used to encode the value of each bit of the 2^n-bit (resp., n-bit) counter.

D. Let $\widetilde{\Sigma} = \Sigma \cup (Q \times \Sigma)$. Let $u \in T$ be a b-node labelled by $check_1$, with $\widetilde{\Sigma}$-label σ, and associated with a TM block bl that does not belong to the first TM configuration. By Properties **B** and **C** above, there is exactly one path π from u that visits a node u' labelled by opt_3. Moreover, u' is followed (along π) from a sequence of nodes labelled by $currBlock$ associated with a TM block bl' with the same cell number as bl and belonging to the precedent TM configuration w.r.t. ν. Also, by Property 2.1 above, the TM block bl_p (if any) following bl' (along π) is labelled by $precBlock$. Let σ_s be the $\widetilde{\Sigma}$-value of node u', and σ' (resp., σ_p) be the $\widetilde{\Sigma}$-value of the TM block bl' (bl_p). We have to require that $\sigma = next_l(\sigma_p, \sigma', \sigma_s)$ if u corresponds to a block of the left successor of the configuration associated with u' (i.e., u is labelled by l), and $\sigma = next_r(\sigma_p, \sigma', \sigma_s)$ otherwise. This requirement can be expressed by the following CTL^* formula φ_D. We distinguish three cases depending on whether u belongs to the first block, to the last block or to a non-extremal block of

the associated TM configuration. For simplicity, we consider only the case in which u belongs to a non-extremal block. The other cases can be handled similarly.

$$
\begin{array}{l}
AG\ \Big[\big(b\ \wedge check_1\ \wedge\ \neg fc\ \wedge\ \neg e\ \wedge\ \ \neg EF(e\ \wedge opt_3)\big) \longrightarrow \\
\quad \Big(\bigvee_{\sigma_1,\sigma_2,\sigma_3 \in \widetilde{\Sigma}}\ E(F(\sigma_1\ \wedge\ opt_3\ \wedge F\ (\sigma_2\ \wedge b\ \wedge\ currBlock\ \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad F(\sigma_3\ \wedge b\ \wedge\ precBlock)))) \bigwedge \\
\qquad\qquad (l\ \rightarrow next_l(\sigma_3,\sigma_2,\sigma_1))\ \bigwedge \\
\qquad\qquad (\neg l\ \rightarrow next_r(\sigma_3,\sigma_2,\sigma_1))\ \Big)\Big]
\end{array}
$$

Therefore, φ_{opt_2} is given by $\varphi_B\ \wedge\ \varphi_{C.1}\ \wedge \varphi_{C.2}\ \wedge\ \varphi_D$. Finally, formula φ is defined in terms of φ_{opt_1} and φ_{opt_2} exactly as for the CTL case. □

References

[1] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *Proc. 8th International Conference on Concurrency Theory (CONCUR'97)*, LNCS 1243, pages 135–150. Springer-Verlag, 1997.

[2] L. Bozzelli. Complexity Results on Branching-Time Pushdown Model Checking. In *Proc. 7th Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'06)*, LNCS 3855, pages 65–79. Springer-Verlag, 2006.

[3] J.R. Buchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pages 1–12, Stanford, 1962.

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

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

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

[7] E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs. In *29th Annual IEEE Symposium on Foundations of Computer Science (FOCS'88)*, pages 328–337, 1988.

[8] E.A. Emerson and C.S. Jutla. Tree automata, μ-calculus and determinacy. In *32nd Annual IEEE Symposium on the Foundations of Computer Science (FOCS'91)*, pages 368–377, 1991.

[9] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[10] O. Kupferman and O. Grumberg. Buy one, get one free!!! *Journal of Logic and Computation*, 6(4):523–539, 1996.

[11] O. Kupferman, N. Piterman, and M.Y. Vardi. Pushdown specifications. In *9th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'02)*, LNAI 2514, pages 262–277. Springer-Verlag, 2002.

[12] O. Kupferman, P.S. Thiagarajan, P. Madhusudan, and M.Y. Vardi. Open systems in reactive environments: Control and Synthesis. In *Proc. 11th International Conference on Concurrency Theory (CONCUR'00)*, LNCS 1877, pages 92–107. Springer-Verlag, 2000.

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

[14] O. Kupferman, M.Y. Vardi, and P. Wolper. Module Checking. *Information and Computation*, 164(2):322–344, 2001.

[15] C. Loding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'04)*, pages 408–420. Springer-Verlag, 2004.

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

[17] J.P. Queille and J. Sifakis. Specification and verification of concurrent programs in Cesar. In *Proceedings of the Fifth International Symposium on Programming*, LNCS 137, pages 337–351. Springer-Verlag, 1981.

[18] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32(2):182–221, 1986.

[19] M.Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, LNCS 1443, pages 628–641. Springer-Verlag, 1998.

[20] I. Walukiewicz. Pushdown processes: Games and Model Checking. In *Proc. 8th International Conference on Computer Aided Verification (CAV'96)*, LNCS 1102, pages 62–74. Springer-Verlag, 1996.

[21] 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)*, LNCS 1974, pages 127–138. Springer-Verlag, 2000.