μ-Calculus Pushdown Module Checking with Imperfect State Information

Benjamin Aminof, Axel Legay, Aniello Murano, Olivier Serre

μ-calculus Pushdown Module Checking with Imperfect State Information

Benjamin Aminof[1], Axel Legay[2], Aniello Murano[3], and Olivier Serre[4]

[1] Hebrew University, Jerusalem 91904, Israel.
[2] University of Liège, Belgium.
[3] Università degli Studi di Napoli "Federico II", 80126 Napoli, Italy.
[4] LIAFA, CNRS & Université Paris VII, France.

Abstract. The model checking problem for open systems (*module checking*) has recently been the subject of extensive study. The problem was first studied by Kupferman, Vardi, and Wolper for finite-state systems and properties expressed in the branching time logics *CTL* and *CTL**. Further study continued mainly in two directions: considering systems equipped with a pushdown store, and considering environments with imperfect information about the system.
A recent paper combined the two directions and considered the *CTL* pushdown module checking problem in the imperfect information setting, i.e., in the case where the environment has only a partial view of the system control states and pushdown store content. It has been shown that this problem is undecidable when the environment has imperfect information about the pushdown store content, while it is decidable and 2Exptime-complete when the imperfect information only concerns control states. It was left open whether the latter remains decidable also for more expressive logics. In this paper, we answer this question in the affirmative, showing that the pushdown module checking problem with imperfect information about the control states is decidable and 2Exptime-complete for the propositional and the graded μ-calculus, and 3Exptime-complete for *CTL**.

1 Introduction

A main distinction in system modeling is between closed systems, whose behavior is totally determined by the program, and open systems, which are systems where the program interacts with an external environment [HP85, Hoa85]. In order to check whether a closed system satisfies a required property we translate the system into a formal model (such as a transition system), specify the property with a temporal-logic formula (such as *CTL* [CE81], *CTL** [EH86], and μ-calculus [Koz83]), and check formally that the model satisfies the formula. This process is called *model checking* ([CE81, QS81]). Checking whether an open system satisfies a required temporal logic formula is much harder, as one has to consider the interaction of the system with all possible environments.

In this paper, we consider open systems which are modeled in the framework introduced by Kupferman, Vardi, and Wolper. Concretely, in [KV96, KVW01], an open finite-state system is described by an extended transition system called a *module*, whose set of states is partitioned into *system states* (where the system makes a transition) and *environment states* (where the environment makes a transition). Given a module $\mathcal{M}$, describing the system to be verified, and a branching time temporal logic formula φ, specifying the desired behavior of the system, the problem of model checking a module, called *module checking*, asks whether for all possible environments, $\mathcal{M}$ satisfies φ. In particular, it might be that the environment does not enable all the external choices. Module checking thus involves not only checking that the full computation tree obtained by unwinding $\mathcal{M}$ (which corresponds to the interaction of $\mathcal{M}$ with a maximal environment) satisfies the specification φ, but also that every tree obtained from it by pruning children of environment nodes (this corresponds to the different choices of different environments) satisfies φ. For example, consider an ATM machine that allows customers to deposit money, withdraw money, check balance, etc. The machine is an open system, and an environment for it is a subset of the set of all possible infinite lines of customers, each with their own plans. Accordingly, there are many different possible environments to consider.

The finite-state system module checking problem, for CTL and CTL^* formulas, has been investigated in [KV96, KVW01]; while for propositional μ-calculus formulas it has been investigated in [FM07]. In all these cases, it has been shown that module checking is exponentially harder than model checking. However, an interesting aspect of these results is that they bear on the corresponding automata-based results for closed systems [KVW00], which gives the hope for practical implementations and applications.

Recently, the module checking idea has been extended to pushdown systems [BMP05], and it has been shown that CTL and μ-calculus pushdown module checking is 2EXPTIME-complete, while CTL^* pushdown module checking is 3EXPTIME-complete [BMP05, FMP07]. Another extension of the module checking idea has been the investigation of environments with *imperfect information*. The first results on the subject were dedicated to finite-state systems [KV97]. In this framework, every state of the module is a composition of *visible* and *invisible* variables, where the latter are hidden from the environment. While a composition of a module $\mathcal{M}$ with an environment with perfect information corresponds to arbitrary disabling of transitions in $\mathcal{M}$, the composition of $\mathcal{M}$ with an environment with imperfect information is such that whenever two computations of the system differ only in the values of invisible variables along them, the disabling of transitions along them coincide. In [KV97] it has been shown that CTL and CTL^* module checking with imperfect information is harder than module checking with perfect information. The results in [KV97] were recently extended in [AMV07] to pushdown systems. In this framework, environments with imperfect information about the system's control state and pushdown store content are considered. Like in the finite-state case, the control states are assignments to Boolean *control variables*, some of which are visible and

some of which are not. Similarly, symbols of the pushdown store are assignments to Boolean visible and invisible *pushdown store variables*. It has been shown in [AMV07] that in the presence of imperfect information, *CTL* pushdown module-checking becomes undecidable, and that the undecidability relies upon hiding information about the pushdown store. Indeed, it was shown that *CTL* pushdown module checking with imperfect state information but visible pushdown store is decidable and 2EXPTIME-complete.

[AMV07] left open the question whether the pushdown module checking problem with imperfect state information, but visible pushdown store, is still decidable when more expressive logics are considered. In this paper we answer this question in the affirmative. Our main contribution is showing that this problem is decidable and 2EXPTIME-complete for the propositional μ-calculus and the graded μ-calculus [KSV02][1], and 3EXPTIME-complete for CTL^*. The lower bound follows from the known perfect information case. For the upper bound we use an automata theoretic approach, and reduce the problem to the emptiness problem of a *semi-alternating pushdown parity tree automaton* (PD-SPT). These are alternating pushdown parity tree automata that behave deterministically on the pushdown store content. That is, two copies of the automaton that read the same input, from two configurations that have the same top of pushdown store, must push the same value into the pushdown store. In this paper, we show that unlike alternating pushdown parity tree automata, for which the emptiness problem is undecidable[2], the emptiness problem for PD-SPT is solvable in 2EXPTIME, which allows us to get the required upper bound for our problem.

2 Preliminaries

In this section, we first recall the concept of open system. Then, we introduce the logics that will be model checked.

2.1 Open Systems.

Let Υ be a finite set. An *Υ-tree* is a prefix closed subset $T \subseteq \Upsilon^*$. The elements of T are called *nodes* and the empty word ε is the *root* of T. For $v \in T$, the set of *children* of v (in T) is $child(T, v) = \{v \cdot x \in T \mid x \in \Upsilon\}$. Given a node $v = u \cdot x$, with $u \in \Upsilon^*$ and $x \in \Upsilon$, we define $last(v)$ to be x. The *complete* Υ-tree is the tree Υ^*. For $v \in T$, a (full) path π of T from v is a *minimal* set $\pi \subseteq T$, such that $v \in \pi$ and for each $v' \in \pi$, such that $child(T, v') \neq \emptyset$, there is exactly one node in $child(T, v')$ belonging to π. Note that every $w \in \Upsilon^\omega$ can

[1] The graded μ-calculus extends the propositional μ-calculus by allowing graded modalities, which enable statements about the number of successors of a state.

[2] Since the emptiness problem of the intersection of two context free languages is undecidable [HU79], the emptiness problem of alternating pushdown automata is undecidable, already in the case of finite words.

be thought of as an infinite path in the tree Υ^*, namely the path containing all the finite prefixes of w. For an alphabet Σ, a Σ-labeled Υ-tree is a pair $\langle T, V\rangle$ where T is an Υ-tree and $V : T \to \Sigma$ maps each node of T to a symbol in Σ.

An *open system* is a system that interacts with its environment and whose behavior depends on this interaction. We consider the case where the environment has imperfect information about the system, i.e., when the system has internal variables that are not visible to its environment. We describe such a system by a *module* $\mathcal{M} = \langle AP, W_s, W_e, w_0, R, L, \cong\rangle$, where AP is a finite set of *atomic propositions*, W_s is a set of *system states*, and W_e is a set of *environment states*. We assume that $W_s \cap W_e = \emptyset$, and call $W = W_s \cup W_e$ the set of $\mathcal{M}$'s *states*. The state $w_0 \in W$ is the *initial state*, $R \subseteq W \times W$ is a total *transition relation*, $L : W \to 2^{AP}$ is a labeling function that maps each state of $\mathcal{M}$ to the set of atomic propositions that hold in it, and $\cong$ is an equivalence relation on W. A module $\mathcal{M}$ is *closed* if $W_e = \emptyset$. States that are indistinguishable by the environment are equivalent according to $\cong$. We write $[W]$ for the set of equivalence classes of W under $\cong$. For the environment, the states of the system are actually the equivalence classes themselves. The equivalence class $[w]$ of a state $w \in W$ is called the *visible part* of w. We write $vis(w)$, instead of $[w]$, to emphasize this.

Given $\langle w, w'\rangle \in R$, w' is a *successor* of w. For each state $w \in W$, we denote by $succ(w)$ the set (possibly empty) of w's successors. A *computation* of $\mathcal{M}$ is a sequence $w_0 \cdot w_1 \cdots$ of states, such that for all $i \geq 0$ we have $\langle w_i, w_{i+1}\rangle \in R$. The set of all (maximal) computations of $\mathcal{M}$ starting from the initial state w_0 can be described by an AP-labeled W-tree $\langle T_{\mathcal{M}}, V_{\mathcal{M}}\rangle$ called a *computation tree*, which is obtained by unwinding $\mathcal{M}$ in the usual way. Each node $v = v_1 \cdots v_k$ of $\langle T_{\mathcal{M}}, V_{\mathcal{M}}\rangle$ describes the (partial) computation $w_0 \cdot v_1 \cdots v_k$ of $\mathcal{M}$, with the root ε corresponding to w_0. The children of v are exactly all nodes of the form $v_1 \cdots v_k \cdot w$, where w ranges over all the successors of v_k in $\mathcal{M}$. We extend the definition of vis to nodes in the natural way. Thus, the visible part of a node v is $vis(v) = vis(v_1) \cdots vis(v_k)$. The labeling $V_{\mathcal{M}}$ of a node v depends on the state it corresponds to (its last state), i.e., $V_{\mathcal{M}}(v) = L(last(v))$. Also, if v corresponds to an environment state we say that v is an *environment node*.

Whenever $\mathcal{M}$ interacts with an environment ξ, its possible moves from environment states (i.e., states in W_e) depend on the behavior of ξ. We can think of an environment to $\mathcal{M}$ as a strategy $\xi : [W]^* \to \{\top, \bot\}$ that maps a finite history s of a computation, as seen by the environment, to either $\top$ or $\bot$, meaning that the environment respectively allows or disallows $\mathcal{M}$ to trace s (obviously, if s is a successor of a system state, the decision whether to trace s or not is made by the system, and we ignore the environment's value of $\xi(s)$). Observe that if an environment disallows $\mathcal{M}$ to trace s, it effectively disallows $\mathcal{M}$ to trace any of the successors of s. Note that one can either require that for every $y \in [W]^*$, if $\xi(x) = \bot$ then $\xi(x \cdot y) = \bot$, or simply ignore the value $\xi(x \cdot y)$. We chose the latter. We say that the tree $\langle [W]^*, \xi\rangle$ maintains the strategy applied by ξ, and we call it a *strategy tree*. We denote by $\mathcal{M} \lhd \xi$ the AP-labeled W-tree induced by the composition of $\langle T_{\mathcal{M}}, V_{\mathcal{M}}\rangle$ with ξ; that is, the AP-labeled W-tree

obtained by pruning from $\langle T_{\mathcal{M}}, V_{\mathcal{M}}\rangle$ subtrees according to ξ. Note that by the definition above, ξ may disable all the children of a node v. Since we usually do not want the environment to completely block the system, we require that at least one child of each node is enabled. In this case, we say that the composition $\mathcal{M} \triangleleft \xi$ is *deadlock-free*. Given a module $\mathcal{M}$, and a strategy tree $\langle [W]^*, \xi\rangle$ for an environment ξ, an AP-labeled W-tree $\langle T, V\rangle$ corresponds to $\mathcal{M} \triangleleft \xi$ iff:

- The root of T corresponds to w_0.
- For $v \in T$ with $last(v) \in W_s$, we have $child(T, v) = \{v \cdot w_1, \ldots, v \cdot w_n\}$, where $succ(last(v)) = \{w_1, \ldots, w_n\}$.
- For $v \in T$ with $last(v) \in W_e$, there exists a nonempty subset $\{w_1, \ldots, w_k\}$ of $succ(last(v))$ such that $child(T, v) = \{v \cdot w_1, \ldots, v \cdot w_k\}$. Furthermore, for all w in $\{w_1, \ldots, w_k\}$ we have that $\xi(vis(v \cdot w)) = \top$, while for all w in $succ(last(v)) \setminus \{w_1, \ldots, w_k\}$ we have that $\xi(vis(x \cdot w)) = \bot$.
- For every node $v \in T$, we have that $V(v) = L(last(v))$.

For a module $\mathcal{M}$ and a temporal logic formula φ defined over AP, we say that $\mathcal{M}$ *reactively satisfies* φ, denoted $\mathcal{M} \models_r \varphi$, if $\mathcal{M} \triangleleft \xi$ satisfies φ, for every environment ξ for which $\mathcal{M} \triangleleft \xi$ is deadlock-free. The problem of deciding whether $\mathcal{M} \models_r \varphi$ is called the *module checking problem with imperfect information*.

2.2 Logics.

In this paper, we consider φ to be either a CTL^* or a propositional/graded μ-calculus formula. The syntax and semantics of CTL^* and μ-calculus are well known, and we assume that the reader is familiar with them (for references, see [Koz83] and [KVW00]). In the rest of this section, we focus on graded μ-calculus, which is an extension of the propositional μ-calculus that allows *graded modalities*. These modalities are denoted by $\langle n\rangle$ ("*exist at least n-successors*") and $[n]$ ("*all but at most n successors*"), respectively.

Formally, we have the following. Let AP and Var be finite and pairwise disjoint sets of *atomic propositions* and *propositional variables*. The set of *graded μ–calculus* formulas is the smallest set such that (i) **true** and **false** are formulas; (ii) p and $\neg p$, for $p \in AP$, are formulas; (iii) $x \in Var$ is a formula; (iv) if φ_1 and φ_2 are formulas, n is a non negative integer, and $y \in Var$, then $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $\langle n\rangle\varphi_1$, $[n]\varphi_1$, $\mu y.\varphi_1(y)$, and $\nu y.\varphi_1(y)$ are also formulas. Observe that we use positive normal form, i.e., negation is applied only to atomic propositions. We often refer to the *graded modalities* $\langle n\rangle\varphi_1$ and $[n]\varphi_1$ as, respectively, *atleast formulas* and *allbut formulas*, and assume that the integers in these operators are given in binary coding: the contribution of n to the length of each of the formulas $\langle n\rangle\varphi$ and $[n]\varphi$ is $\lceil \log n \rceil$, rather than n.

The definition of the semantics of graded μ-calculus w.r.t an AP-labeled W-tree $\langle T, V\rangle$ is similar to that of the standard μ-calculus, except for the graded modalities. Informally, an *atleast* formula $\langle n\rangle\varphi$ holds at a node w of the tree if φ holds in at least $n + 1$ children of the node. Dually, the *allbut* formula $[n]\varphi$ holds in a node of the tree $\mathcal{K}$ if φ holds in all but at most n of its successors.

Due to space limitation, we refer the reader to [KSV02] (also [BLMV06]) for a formal description of the full semantics.

3 Imperfect Information Pushdown Module Checking

In this section, we consider infinite-state modules which are induced by *Open Pushdown Systems* (*OPD*) [AMV07]. In our framework, the environment has imperfect information about the internal control states of the system, but the pushdown store is visible.

Definition 1. *An* OPD *is a tuple* $\mathcal{S} = \langle AP, Q, q_0, \Gamma, \flat, \delta, \eta, Env \rangle$*, where* AP *is a finite set of atomic propositions;* Q *is a finite set of (*control*)* states*; and* $q_0 \in Q$ *is an* initial state*. We assume that* $Q \subseteq 2^{V \cup H}$*, where* V *and* H *are disjoint finite sets of* visible *and* invisible control variables*, respectively.* Γ *is a finite pushdown store alphabet;* $\flat \notin \Gamma$ *is the* pushdown store bottom symbol*, and we use* $\Gamma_\flat$ *to denote* $\Gamma \cup \{\flat\}$*. The transition relation* $\delta \subseteq (Q \times \Gamma_\flat) \times (Q \times \Gamma_\flat^*)$ *is finite;* $\eta : Q \times \Gamma_\flat \to 2^{AP}$ *is a labeling function; and* $Env \subseteq Q \times \Gamma_\flat$ *is used to specify the set of* environment configurations*. The* size $|\mathcal{S}|$ *of* $\mathcal{S}$ *is* $|Q| + |\Gamma| + |\delta|$*, with* $|\delta| = \sum_{((p,\gamma),(q,\beta)) \in \delta} |\beta|$*.*

A *configuration* of $\mathcal{S}$ is a pair (q, α), where q is a control state and $\alpha \in \Gamma^* \cdot \flat$ is a pushdown store content. We write $top(\alpha)$ for the leftmost symbol of α, and call it the *top of the pushdown store* α. The *OPD* moves according to the transition relation. Thus, $((p, \gamma), (q, \beta)) \in \delta$ implies that if the *OPD* is in state p, and the top of the pushdown store is γ, then it can move to state q, pop γ and push β. We assume that $\flat$ is always present at the bottom of the pushdown store, and nowhere else. Note that we make this assumption also about the various pushdown automata we use later. For a control state $q \in Q$, the visible part of q is $vis(q) = q \cap V$. The visible part of a configuration (q, α), is thus $vis((q, \alpha)) = (vis(q), \alpha)$. As for modules, the designation of a configuration of an *OPD* as an environment configuration is known to the environment. Thus, we require that for every two configurations (q, α) and (q', α'), such that $vis(q) = vis(q')$, it holds that $(q, top(\alpha)) \in Env$ iff $(q', top(\alpha')) \in Env$.

Definition 2. *An* OPD $\mathcal{S} = \langle AP, Q, q_0, \Gamma, \flat, \delta, \eta, Env \rangle$ *induces an infinite-state module* $\mathcal{M}_S = \langle AP, W_s, W_e, w_0, R, L, \cong \rangle$*, possibly with invisible information, where* AP *is a set of atomic propositions;* $W_s \cup W_e = Q \times \Gamma^* \cdot \flat$ *is the set of configurations;* W_e *is the set of configurations* (q, α) *such that* $(q, top(\alpha)) \in Env$*;* $w_0 = (q_0, \flat)$ *is the initial configuration;* R *is a transition relation, where* $((q, \gamma \cdot \alpha), (q', \beta)) \in R$ iff *there exist* $((q, \gamma), (q', \beta')) \in \delta$ *such that* $\beta = \beta' \cdot \alpha$*;* $L((q, \alpha)) = \eta(q, top(\alpha))$ *for all* $(q, \alpha) \in W$*; and for every two configurations* $w, w' \in W$*, we have that* $w \cong w'$ *iff* $vis(w) = vis(w')$*.*

To describe the interaction of an *OPD* $\mathcal{S}$ with its environment we consider the interaction of the environment with the induced module $\mathcal{M}_S$. Indeed, every

environment ξ of $\mathcal{S}$ can be represented by a strategy tree $\langle [W]^*, \xi \rangle$, and the composition $\mathcal{M}_S \lhd \xi$ of $\langle [W]^*, \xi \rangle$ with $\langle T_{\mathcal{M}_S}, V_{\mathcal{M}_S} \rangle$ describes all the computations of $\mathcal{S}$ allowed by the environment ξ.

We consider the *pushdown module checking problem with imperfect state information*, i.e., given an *OPD* $\mathcal{S}$ and a formula φ, decide whether $\mathcal{M}_S \models_r \varphi$.

The pushdown module checking problem with imperfect state information is known to be 2EXPTIME-complete when φ is a *CTL* formula [AMV07]. In this paper, we answer an open question of [AMV07] and show that the problem remains 2EXPTIME-complete when considering φ to be a propositional or a graded μ-calculus formula, and that it becomes 3EXPTIME-complete when φ is a *CTL** formula. For the upper bound, we reduce our problem to the emptiness problem of a semi-alternating pushdown parity tree automata.

3.1 Semi-Alternating Pushdown Tree Automata.

We start with the definition of *semi-alternating pushdown parity tree automata* (*PD-SPT*), first introduced in [AMV07] w.r.t. a Büchi acceptance condition. A PD-SPT is a tuple $\mathcal{A} = \langle \Sigma, D, \Gamma, Q, q_0, \flat, \delta, F \rangle$, where Σ is a finite input alphabet, D is a finite set of *directions*, Γ is a finite pushdown store alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\flat \notin \Gamma$ is the pushdown store bottom symbol, and F is a parity acceptance condition (to be defined later). Moreover, δ is a finite transition relation defined as a function $\delta : Q \times \Sigma \times \Gamma_\flat \to \mathcal{B}^+(D \times Q \times \Gamma_\flat^*)$, where, as usual, $\Gamma_\flat = \Gamma \cup \{\flat\}$, and $\mathcal{B}^+(D \times Q \times \Gamma_\flat^*)$ is the set of all finite positive Boolean combinations of triples (d, q, β), where d is a direction, q is a state, and β is a word made of pushdown store symbols. We also allow the formulas **true** and **false**. We write $S \in \delta(p, \sigma, \gamma)$ to denote that S is a set of tuples (d, q, β) that satisfy $\delta(p, \sigma, \gamma)$.

What makes the automaton semi-alternating is the requirement that for every $d \in D$, $\sigma \in \Sigma$, $p, p' \in Q$ (possibly the same state), and $\gamma \in \Gamma$, if (d, q, β) appears in $\delta(p, \sigma, \gamma)$, and (d, q', β') appears in $\delta(p', \sigma, \gamma)$, then $\beta = \beta'$. That is, two copies of the automaton that read the same input, from two configurations that have the same top symbol of the pushdown store, and proceed in the same direction, must push the same value into the pushdown store. In particular, it follows that in every run, two copies of the automaton that are reading the same node of an input tree have the same pushdown store content. Note that if we remove the semi-alternation requirement the resulting automaton is called *alternating pushdown parity tree automaton* (*PD-APT*).

As a special case of PD-APT, we consider *nondeterministic pushdown parity tree automata* (*PD-NPT*), where the concurrency feature (i.e., the $\wedge$ operator in δ) is not allowed. That is, whenever a PD-NPT visits a node x of the input tree, it sends to each successor (direction) of x at most one copy of itself. More formally, a PD-NPT is a PD-APT in which δ is in disjunctive normal form, and in each conjunct each direction appears at most once. Note that if $\mathcal{A}$ is a PD-APT with $\Gamma = \emptyset$, its pushdown store is neutralized, hence, $\mathcal{A}$ is simply called an *alternating parity tree automaton* (*APT*), and we can abbreviate and write

$\mathcal{A} = \langle \Sigma, D, Q, q_0, \delta, F\rangle$, where $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$. Similarly, a PD-NPT with an empty pushdown store alphabet is called a *nondeterministic parity tree automaton* (*NPT*).

A run of a PD-SPT $\mathcal{A}$, on a Σ-labeled tree $\langle T, V\rangle$, with $T = D^*$, is a $(D^* \times Q \times \Gamma^* \cdot \flat)$-labeled $\mathbb{N}$-tree $\langle T_r, r\rangle$, such that the root is labeled with $(\varepsilon, q_0, \flat)$ and the labels of each node and its successors satisfy the transition relation. Formally, a $(D^* \times Q \times \Gamma^* \cdot \flat)$-labeled tree $\langle T_r, r\rangle$ is a run of $\mathcal{A}$ on $\langle T, V\rangle$ iff

- $r(\varepsilon) = (\varepsilon, q_0, \flat)$, and
- for all $x \in T_r$ such that $r(x) = (y, p, \gamma \cdot \alpha)$, there is an $n \in \mathbb{N}$ such that the successors of x are exactly $x{\cdot}1, \ldots x{\cdot}n$, and for all $1 \leq i \leq n$ we have $r(x{\cdot}i) = (y \cdot d_i, p_i, \beta_i \cdot \alpha)$ for some $\{(d_1, p_1, \beta_1), \ldots, (d_n, p_n, \beta_n)\} \in \delta(p, V(y), \gamma)$.

For a path $\pi \subseteq T_r$, let $inf_r(\pi) \subseteq Q$ be the set of states that appear in the labels of infinitely many nodes in π. For a parity condition $F = \{F_1, F_2, \ldots, F_k\}$, with $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_k = Q$, we have that π is *accepting* iff the minimal index i, for which $inf_r(\pi) \cap F_i \neq \emptyset$, is even. The number k is called the *index* of the automaton. A run $\langle T_r, r\rangle$ is *accepting* iff all its paths are accepting. The automaton $\mathcal{A}$ accepts an input tree $\langle T, V\rangle$ iff there is an accepting run of $\mathcal{A}$ on $\langle T, V\rangle$. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of Σ-labeled trees with branching degree D accepted by $\mathcal{A}$. We say that an automaton $\mathcal{A}$ is nonempty iff $L(\mathcal{A}) \neq \emptyset$. Given a PD-SPT $\mathcal{A} = \langle \Sigma, D, \Gamma, Q, q_0, \flat, \delta, F\rangle$, we define the size of δ as the sum of the lengths of the satisfiable (i.e., not **false**) formulas that appear in $\delta(q, \sigma, \gamma)$, for some q, σ, and γ.

3.2 Simulating a PD-SPT by a PD-NPT.

As mentioned in [AMV07], alternating pushdown automata are not equivalent to nondeterministic ones. However, as we show here, the limitations imposed on semi-alternating automata allow us to translate a PD-SPT to an equivalent PD-NPT[3]. A key observation is that since a pushdown store operation performed by a semi-alternating automaton does not depend on the current (or next) control states, we can split the transition function of a PD-SPT into two functions: a *state transition function* δ_Q, and a *pushdown store update function* δ_Γ, as follows. Given a PD-SPT $\mathcal{A} = \langle \Sigma, D, \Gamma, Q, q_0, \flat, \delta, F\rangle$, let $\delta_Q : Q \times \Sigma \times \Gamma_\flat \to \mathcal{B}^+(D \times Q)$ be the projection of δ on $\mathcal{B}^+(D \times Q)$. That is, $\delta_Q(q, \sigma, \gamma)$ is obtained from $\delta(q, \sigma, \gamma)$ by replacing every element (d, q, β) that appears in $\delta(q, \sigma, \gamma)$ with (d, q). The pushdown store update function $\delta_\Gamma : \Sigma \times \Gamma_\flat \times D \to \Gamma_\flat^*$, is a partial function; for every $(p, \sigma, \gamma) \in Q \times \Sigma \times \Gamma_\flat$ and every $(d, q, \beta) \in D \times Q \times \Gamma_\flat^*$, such that (d, q, β) appears in $\delta(p, \sigma, \gamma)$, we let $\delta_\Gamma(\sigma, \gamma, d) = \beta$. Since $\mathcal{A}$ is semi-alternating, δ_Γ is well defined. Observe that for every $(p, \sigma, \gamma) \in Q \times \Sigma \times \Gamma_\flat$ we

[3] The translation used in [AMV07], for semi-alternating pushdown Büchi tree automata, made a crucial use of the Büchi acceptance condition, and can not be extended to the parity acceptance condition.

have that $\delta(p,\sigma,\gamma)$ can be obtained from $\delta_Q(p,\sigma,\gamma)$ by replacing every (d,q) that appears in $\delta_Q(p,\sigma,\gamma)$ with $(d,q,\delta_\Gamma(\sigma,\gamma,d))$.

Consider a Σ-labeled tree $\langle T,V\rangle$, with $T=D^*$. Note that for every node $x\in T$ and every run of $\mathcal{A}$ on $\langle T,V\rangle$, the pushdown store content of all the copies of $\mathcal{A}$ that visit x is the same, and only depends on x. We can thus define a function $\Delta_\Gamma : T\to\Gamma_\flat^*$, giving for every node x its associated pushdown store content, as follows: (1) $\Delta_\Gamma(\varepsilon)=\flat$, and (2) for all $x\cdot d\in T$ we have $\Delta_\Gamma(x\cdot d)=\delta_\Gamma(V(x),\gamma,d)\cdot\beta$, where $\Delta_\Gamma(x)=\gamma\cdot\beta$, and $\gamma\in\Gamma_\flat$.

Annotating input trees with pushdown store symbols enables us to simulate a PD-SPT by an APT running on the annotated version of an input tree. Given a Σ-labeled tree $\langle T,V\rangle$, we define its *$\Gamma_\mathcal{A}$-annotation* to be the $(\Sigma\times\Gamma_\flat)$-labeled tree $\langle T,U\rangle$, obtained by letting $U(x)=(V(x),top(\Delta_\Gamma(x)))$, for every $x\in T$.

Lemma 1. *Let $\mathcal{A}=\langle\Sigma,D,\Gamma,Q,q_0,\flat,\delta,F\rangle$ be a PD-SPT. There is an APT $\tilde{\mathcal{A}}$, such that $\mathcal{A}$ accepts $\langle T,V\rangle$ iff $\tilde{\mathcal{A}}$ accepts the $\Gamma_\mathcal{A}$-annotation of $\langle T,V\rangle$.*

Proof. Consider the APT $\tilde{\mathcal{A}}=\langle\Sigma\times\Gamma_\flat,D,Q,q_0,\tilde{\delta},F\rangle$, where $\tilde{\delta}(q,(\sigma,\gamma))=\delta_Q(q,\sigma,\gamma)$. It is not hard to see that every run $r=\langle T_r,r\rangle$ of $\mathcal{A}$ on $\langle T,V\rangle$ induces a corresponding run $r'=\langle T_r,r'\rangle$ of $\tilde{\mathcal{A}}$ on the $\Gamma_\mathcal{A}$-annotation of $\langle T,V\rangle$, and vice versa. The connection between r and r' being that for every $x\in T_r$, we have that $r(x)=(y,p,\alpha)$ iff $r'(x)=(y,p)$ and $\Delta_\Gamma(x)=\alpha$. □

By [MS87], every APT can be translated to an equivalent NPT. Hence, Lemma 1 implies that if $\mathcal{A}$ is a PD-SPT, then there is an NPT $\mathcal{A}'$ such that $\mathcal{A}$ accepts $\langle T,V\rangle$ iff $\mathcal{A}'$ accepts the $\Gamma_\mathcal{A}$-annotation of $\langle T,V\rangle$. This allows us to translate $\mathcal{A}$ to an equivalent PD-NPT $\mathcal{A}''$ (running on the same input trees as $\mathcal{A}$). Given a Σ-labeled tree, $\mathcal{A}''$ generates on the fly its $\Gamma_\mathcal{A}$-annotation and runs A' on the annotated tree. Formally, we have the following:

Theorem 1. *Every PD-SPT can be translated to an equivalent PD-NPT.*

Proof. Let $\mathcal{A}=\langle\Sigma,D,\Gamma,Q,q_0,\flat,\delta,F\rangle$ be a PD-SPT and $\tilde{\mathcal{A}}=\langle\Sigma\times\Gamma_\flat,D,Q,q_0,\tilde{\delta},F\rangle$ be an APT derived from $\mathcal{A}$ by Lemma 1. By [MS87], $\tilde{\mathcal{A}}$ has an equivalent NPT $\mathcal{A}'=\langle\Sigma\times\Gamma_\flat,D,Q',q_0',\delta',F'\rangle$. Consider the PD-NPT $A''=\langle\Sigma,D,\Gamma,Q',q_0',\flat,\delta'',F'\rangle$, where for every $(p,\sigma,\gamma)\in Q'\times\Sigma\times\Gamma_\flat$, we have that $\delta''(p,\sigma,\gamma)$ is obtained from $\delta'(p,(\sigma,\gamma))$ by replacing every (d,q) that appears in $\delta'(p,(\sigma,\gamma))$, with $(d,q,\delta_\Gamma(\sigma,\gamma,d))$. Since $\mathcal{A}'$ is nondeterministic, so is $\mathcal{A}''$. Given a Σ-labeled tree $\langle T,V\rangle$, it is not hard to see that for every $x\in T$, the pushdown store of every copy of $\mathcal{A}''$ that visits x contains exactly $\Delta_\Gamma(x)$. Hence, $\mathcal{A}''$ accepts $\langle T,V\rangle$ iff $\mathcal{A}'$ accepts the $\Gamma_\mathcal{A}$-annotation of $\langle T,V\rangle$, i.e., iff $\mathcal{A}$ accepts $\langle T,V\rangle$. □

3.3 The Emptiness Problem of PD-SPT.

Looking at the automata transformations performed in Theorem 1 and Lemma 1 we see that the only transformation that incurs a blowup in the size of the automaton is the transformation of the APT $\tilde{\mathcal{A}}$ to the NPT $\mathcal{A}'$. By [MS87],

given an APT with n states and index k, running over D^* trees, one can build an equivalent NPT with $(nk)^{O(nk)}$ states, an $O(nk)$ index, and a transition relation of size $(nk)^{O(|D|nk)}$. Hence, starting with a PD-SPT $\mathcal{A}$ with n states and index k, our algorithm yields an equivalent PD-NPT $\mathcal{A}''$ with $(nk)^{O(nk)}$ states, an $O(nk)$ index, and a transition relation of size $(nk)^{O(|D|nk)}$. It is worth noting that the blowup is independent of the size of the transition relation of $\mathcal{A}$. By [KPV02], the emptiness of $\mathcal{A}''$ can be decided in time exponential in the product of the number of states, the index, and the size of the transition relation of $\mathcal{A}''$. Overall, we get the following corollary:

Corollary 1. *The emptiness problem for a PD-SPT with n states and index k, running on D^* trees, can be solved in time double-exponential in $|D|nk$.*

4 Solving Pushdown Module Checking with Imperfect State Information

We first show that the pushdown module checking problem with imperfect state information, for μ-calculus, graded μ-calculus, and CTL^*, can be reduced to the emptiness problem of PD-SPT.

Basically, we extend the automata theoretic approach used in [AMV07] for CTL pushdown module checking with imperfect state information. Before presenting the formal reduction, let us briefly recap the approach taken by [AMV07], and discuss the main changes required to adapt it to the problem we address. Given an OPD $\mathcal{S}$, and a CTL formula φ, one builds an automaton $\mathcal{A}_{\mathcal{S},\varphi}$ that accepts $\{\top,\bot\}$-labeled trees corresponding to strategies ξ, whose composition with $\mathcal{M}_S$ is deadlock-free and satisfies φ. Intuitively, a run of $\mathcal{A}_{\mathcal{S},\varphi}$ on an input strategy tree ξ proceeds by simulating an unwinding of the module $\mathcal{M}_S$, pruned at each step accordingly to the strategy ξ; copies of the automaton which simulate nodes in the computation tree of $\mathcal{M}_S$ that are indistinguishable by the environment are sent to the same direction in the input tree. The resulting run tree of $\mathcal{A}_{\mathcal{S},\varphi}$ on ξ is basically a replica of the composition $\mathcal{M}_S \lhd \xi$, and the fact that it satisfies the formula φ is checked on the fly, by employing in $\mathcal{A}_{\mathcal{S},\varphi}$ the classical alternating-automata approach for model checking CTL.

When considering CTL^* or μ-calculus, adapting the construction used in [AMV07] basically amounts to replacing the embedded alternating automaton that does the on-the-fly model checking: instead of using an automaton that handles CTL, one uses an automaton that handles CTL^* (or μ-calculus). Since an alternating automaton that does μ-calculus model checking is linear in the size of the formula, while one that does CTL^* model checking is exponential in the size of the formula [KVW00], the automaton $\mathcal{A}_{\mathcal{S},\varphi}$ has $O(|\mathcal{S}| * |\varphi|)$ states in the case of μ-calculus, and $O(|\mathcal{S}| * 2^{|\varphi|})$ states in the case of CTL^*. It is important to note that the acceptance condition of $\mathcal{A}_{\mathcal{S},\varphi}$ is essentially that of the embedded model checking automaton. Hence, unlike in [AMV07], where a Büchi condition was enough, for the more expressive logics that we consider

here, we need a stronger acceptance condition, namely, a parity condition, for which solving the emptiness problem requires stronger machinery.

The extension of the construction used in [AMV07] is slightly more delicate when considering graded μ-calculus. Given a graded μ-calculus formula φ, one possible approach is to translate φ into an equivalent μ-calculus formula (without graded modalities). Essentially, as pointed out in [KSV02], one introduces new atomic propositions $p_1, \ldots, p_b$, (where b is the largest number used in the graded modalities in φ) and replaces every atleast formula $\langle n \rangle \psi$ by $\bigvee_{\{i_1,\ldots,i_{n+1}\} \subseteq \{1,\ldots,b\}} \bigwedge_{1 \leq j \leq n+1} \langle 0 \rangle (\psi \wedge p_{i_j})$, and dually for allbut formulas. One also has to conjoin φ with a formula stating that exactly one of the $p_1, \ldots, p_b$ holds at each state, and that successors that are labeled with the same p_i agree on their label with respect to all the formulas in the closure of φ. Unfortunately, since the numbers in the graded modalities are coded in binary, such a translation may result in a μ-calculus formula which is exponentially larger than φ; resulting in an overall exponentially worse complexity for the graded μ-calculus, compared to the un-graded one. In order to avoid this extra exponent, in the context of satisfiability, [KSV02] introduced graded automata. However, graded automata do not transfer directly to the imperfect information setting. Fortunately, there is another solution. Instead of expanding the graded modalities at the formula stage, as suggested above, we can expand them as we build the transition relation of $\mathcal{A}_{\mathcal{S},\varphi}$. Thus, for example, the transition relation of $\mathcal{A}_{\mathcal{S},\varphi}$ will specify that a copy of the automaton, that is responsible for verifying that an atleast formula $\langle n \rangle \psi$ holds at a certain configuration of the OPD, should send $n+1$ copies of itself to one of the exponentially many possible subsets of $n+1$ successors of the current configuration. This expansion of the graded modalities allows $\mathcal{A}_{\mathcal{S},\varphi}$ to handle graded μ-calculus formulas using an embedded regular μ-calculus model checker (without graded modalities). This comes at the price of $\mathcal{A}_{\mathcal{S},\varphi}$ having an exponentially larger transition relation than if graded modalities were not present; but does not affect the number of states, or the index, of $\mathcal{A}_{\mathcal{S},\varphi}$. Since our algorithm for checking the emptiness of PD-SPT is such that its complexity does not depend on the size of the transition relation of the PD-SPT, we handle graded μ-calculus formulas with the same complexity as we do regular μ-calculus formulas.

Theorem 2. *Consider an OPD $\mathcal{S}$ and a propositional, or a graded, μ-calculus (resp.* CTL**) formula φ, over $\mathcal{S}$'s atomic propositions. There is a PD-SPT $\mathcal{A}_{\mathcal{S},\varphi}$ with $O(|\mathcal{S}| * |\varphi|)$ states (resp. $O(|\mathcal{S}| * 2^{|\varphi|})$), and an index $O(|\varphi|)$, such that $L(A_{\mathcal{S},\varphi})$ is exactly the set of strategies ξ for which $\mathcal{M}_S \lhd \xi$ is deadlock-free and satisfies φ.*

Proof (Sketch). We give the construction of $\mathcal{A}_{\mathcal{S},\varphi}$ for the graded μ-calculus. The construction for the propositional μ-calculus is very similar, and the one for *CTL** is obtained by replacing the embedded classical alternating-automata model checker with a *CTL** one.

We first give some extra definitions regarding graded μ-calculus. From now on, we refer to μ and ν as *fixpoint operators*. A propositional variable y occurs

free in a formula if it is not in the scope of a fixpoint operator, and *bounded* otherwise. We use λ to denote a fixpoint operator μ or ν. For a formula $\lambda y.\varphi(y)$, we write $\varphi(\lambda y.\varphi(y))$ to denote the formula that is obtained from $\lambda y.\varphi(y)$ by one-step unfolding; i.e., $\varphi(\lambda y.\varphi(y))$ is obtained by replacing each free occurrence of y in φ with $\lambda y.\varphi(y)$. For technical convenience, we restrict our attention to formulas without free variables (also called *sentences*). The closure $cl(\varphi)$ of a graded μ-calculus sentence φ is the smallest set of graded μ-calculus formulas that contains φ and is closed under sub-formulas (that is, if ψ is in the closure, then so do all its sub-formulas that are sentences) and fixpoint applications (that is, if $\lambda y.\varphi(y)$ is in the closure, then so is $\varphi(\lambda y.\varphi(y))$). As proved in [BLMV06], for every graded μ-calculus formula φ, the number of elements in $cl(\varphi)$ is linear in the length of φ. Accordingly, we define the size $|\varphi|$ of φ to be the number of elements in $cl(\varphi)$.

Let $\mathcal{S} = \langle AP, Q, q_0, \Gamma, \flat, \delta, \eta, Env\rangle$ be an OPD, let φ be a graded μ-calculus formula (guarded[4], without free variables, and in positive normal form), and let $\mathcal{M}_S = \langle AP, W_s, W_e, w_0, R, L, \cong\rangle$ be the module induced by $\mathcal{S}$. We build an automaton $\mathcal{A}_{\mathcal{S},\varphi}$ that accepts $\{\top,\bot\}$-labeled trees corresponding to strategies ξ, whose composition with $\mathcal{M}_S$ is deadlock-free and satisfy φ. Intuitively, a run of $\mathcal{A}_{\mathcal{S},\varphi}$ on an input strategy tree ξ, proceeds by simulating an unwinding of the module $\mathcal{M}_S$, pruned at each step according to the strategy ξ; copies of the automaton simulating nodes in the computation tree of $\mathcal{M}_S$ that are indistinguishable by the environment are sent to the same direction in the input tree. The resulting run tree of $\mathcal{A}_{\mathcal{S},\varphi}$ on ξ is essentially a replica of the composition $\mathcal{M}_S \lhd \xi$, and the fact that it satisfies the formula φ is checked on the fly by employing in $\mathcal{A}_{\mathcal{S},\varphi}$ the usual alternating-automata approach for μ-calculus model checking. In the full computation tree of $\mathcal{M}_S$, the set of directions is $G = \{(q,\beta) \mid ((p,\alpha),(q,\beta)) \in R \text{ for some } p,\alpha \text{ and } \beta\}$. Since in $\mathcal{S}$ the pushdown store is completely visible to the environment, the set of directions of the input strategy trees is $D = \{(vis(q),\beta) \mid ((p,\alpha),(q,\beta)) \in R \text{ for some } p,q,\alpha \text{ and } \beta\}$.

Finally, due to the fact that all copies of the automaton sent to direction $(vis(q),\beta)$ push β into the pushdown store, the resulting automaton $\mathcal{A}_{\mathcal{S},\varphi}$ is semi-alternating. As in [KVW00] we are going to use a function split to avoid the problem of having states with a component in $cl(\varphi)$ that is a disjunction or a conjunction. Without the use of split, a run of the automaton may have no states that correspond to a fixpoint sub-formula of φ that is part of a conjunction or

[4] A graded μ-calculus formula is *guarded* if for every variable y, all the occurrences of y that are in a scope of a fixpoint modality λ are also in the scope of a graded modality that is itself in the scope of λ. For example, the formula $\nu y.(p \vee [0]y)$ is guarded, but the formula $\nu y.(p \vee y)$ is not. Given a graded μ-calculus formula, we can construct in linear time an equivalent guarded formula (see [KVW00] for a proof for μ-calculus, which is easily extendible to the graded setting). Accordingly, we assume that all formulas are guarded. This is essential for the correctness of our construction (it guarantees that transitions involving fixpoint formulas are well defined).

a disjunction, which makes it impossible to correctly define the acceptance condition.

We formally define $\mathcal{A}_{\mathcal{S},\varphi} = \langle \{\top, \bot\}, D, \Gamma, Q', q'_0, \flat, \delta', F \rangle$, where

- $Q' = (Q \times (cl(\varphi) \cup \{p_\top\}) \times \{\forall, \exists\} \times \{p_e, p_s\}) \cup \{q'_0\}$. States with the component $p_\top$ are used to check that the composition of $\mathcal{M}_S$ with the strategy is deadlock-free, while states with a component in $cl(\varphi)$ check that this composition satisfies φ. The components p_e and p_s are used to flag that the currently simulated node, of the computation tree of $\mathcal{M}_S$, is a child of an environment or a system node, respectively. Clearly, the simulation should respect the strategy pruning specifications only if they correspond to children of environment nodes; that is, only if the current state q contains p_e. Every state is either in an existential or a universal mode, as specified by the $\forall$ and $\exists$ components. When the automaton is in a universal state $(q, \varphi, \forall, p_e)$ with a pushdown store content α, it accepts all strategies for which (q, α) in $\mathcal{M}_S$ is either pruned or satisfies φ (where $p_\top$ is satisfied iff the root of the strategy is labeled $\top$). When the automaton is in an existential state $(q, \varphi, \exists, p_e)$ with a pushdown store content α, it accepts all strategies for which (q, α) in $\mathcal{M}_S$ is not pruned and satisfies φ.
- δ' is a function $\delta' : Q' \times \Sigma \times \Gamma_\flat \to \mathcal{B}^+(D \times Q' \times \Gamma_\flat^*)$. Before giving the formal definition, we show an example. Consider, a transition from the configuration $(\langle p, \forall X\psi, \exists, p_e \rangle, \gamma \cdot \alpha)$, where $(p, \gamma) \in Env$. First, if the transition to $(p, \gamma \cdot \alpha)$ is disabled (that is, the automaton reads $\bot$), then, as the current mode is existential, the run is rejecting. If the transition to $(p, \gamma \cdot \alpha)$ is enabled, then the successors of $(p, \gamma \cdot \alpha)$ that are enabled should satisfy ψ. Note that all the successors of $(p, \gamma \cdot \alpha)$ that are indistinguishable by the environment are sent by the automaton to the same direction v. This guarantees that either all these successors are enabled by the strategy (in case the letter to be read in direction v is $\top$) or all are disabled (in case the letter in direction v is $\bot$). In addition, since the requirement to satisfy ψ concerns only successors of $(p, \gamma \cdot \alpha)$ that are enabled, the mode of the new states is universal. The copies of $\mathcal{A}_{\mathcal{S},\varphi}$ that check the composition with the strategy to be deadlock-free guarantee that at least one successor of $(p, \gamma \cdot \alpha)$ is enabled. As noted earlier, the enable/disable instructions of the strategy are ignored in every configuration $(p, \gamma \cdot \alpha)$ that is a successor of a system configuration. Also note that since we assume that no configuration in $\mathcal{M}_S$ has no successors, the conjunctions and disjunctions in δ' cannot be empty.

  We now formally define the transition function δ'. For $(p, \gamma \cdot \alpha) \in W$, we define the set of successors of $(p, \gamma \cdot \alpha)$ in $\mathcal{M}_S$, to be $s(p, \gamma) = \{(q, \beta) \mid ((p, \gamma), (q, \beta)) \in \delta\}$. The transition function $\delta' : Q' \times \Sigma \times \Gamma_\flat \to \mathcal{B}^+(D \times Q' \times \Gamma_\flat^*)$ is defined as follows. In the rules below, for the sake of succinctness, we consider $m \in \{\exists, \forall\} \times \{p_e, p_s\}$, $h \in AP \cup \{\mathbf{true}, \mathbf{false}\}$. Also, given a transition from $(\langle p, \psi, m \rangle, \top, \gamma)$, we let $p_x = p_e$ if $(p, \gamma) \in Env$ and $p_x = p_s$, otherwise.
  - $\delta'(q'_0, \bot, \flat) = \mathbf{false}$ and
    $\delta'(q'_0, \top, \flat) = \delta'(\langle q_0, p_\top, \exists, p_s \rangle, \top, \flat) \wedge \delta'(\langle q_0, \varphi, \exists, p_s \rangle, \top, \flat)$.

- For all p, ψ, and γ, we have $\delta'(\langle p, \psi, \forall, p_e\rangle, \bot, \gamma) = \mathbf{true}$ and $\delta'(\langle p, \psi, \exists, p_e\rangle, \bot, \gamma) = \mathbf{false}$.
- For all p, ψ, and γ, we have $\delta'(\langle p, \psi, \forall, p_s\rangle, \bot, \gamma) = \delta'(\langle p, \psi, \forall, p_s\rangle, \top, \gamma)$ and $\delta'(\langle p, \psi, \exists, p_s\rangle, \bot, \gamma) = \delta'(\langle p, \psi, \exists, p_s\rangle, \top, \gamma)$.
- $\delta'(\langle p, p_\top, m\rangle, \top, \gamma) = (\bigvee_{(q,\beta)\in s(p,\gamma)}(vis(q,\beta), \langle q, p_\top, \exists, p_x\rangle, \beta))\wedge (\bigwedge_{(q,\beta)\in s(p,\gamma)}(vis(q,\beta), \langle q, p_\top, \forall, p_x\rangle, \beta))$.
- $\delta'(\langle p, h, m\rangle, \top, \gamma) = \mathbf{true}$ if $h \in \eta((p,\gamma))$, or $h = \mathbf{true}$.
- $\delta'(\langle p, h, m\rangle, \top, \gamma) = \mathbf{false}$ if $h \notin \eta((p,\gamma))$, or $h = \mathbf{false}$.
- $\delta'(\langle p, \neg h, m\rangle, \top, \gamma) = \mathbf{true}$ if $h \notin \eta((p,\gamma))$, or $h = \mathbf{false}$.
- $\delta'(\langle p, \neg h, m\rangle, \top, \gamma) = \mathbf{false}$ if $h \in \eta((p,\gamma))$, or $h = \mathbf{true}$.
- $\delta'(\langle p, \psi_1 \wedge \psi_2, m\rangle, \top, \gamma) = \mathsf{split}(\delta'(\langle p, \psi_1, m\rangle, \top, \gamma) \wedge \delta'(\langle p, \psi_2, m\rangle, \top, \gamma))$.
- $\delta'(\langle p, \psi_1 \vee \psi_2, m\rangle, \top, \gamma) = \mathsf{split}(\delta'(\langle p, \psi_1, m\rangle, \top, \gamma) \vee \delta'(\langle p, \psi_2, m\rangle, \top, \gamma))$.
- $\delta'(\langle p, [n]\psi, m\rangle, \top, \gamma) = \mathsf{split}(\bigvee_{Y\subseteq s(p,\gamma)\wedge|Y|=|s(p,\gamma)|-n} \bigwedge_{(q,\beta)\in Y}(vis(q,\beta), \langle q, \psi, \forall, p_x\rangle, \beta))$.
- $\delta'(\langle p, \langle n\rangle\psi, m\rangle, \top, \gamma) = \mathsf{split}(\bigvee_{Y\subseteq s(p,\gamma)\wedge|Y|=n+1} \bigwedge_{(q,\beta)\in Y}(vis(q,\beta), \langle q, \psi, \exists, p_x\rangle, \beta))$.
- $\delta'(\langle p, \mu y.\varphi(y), m\rangle, \top, \gamma) = \mathsf{split}(\delta'(\langle p, \varphi(\mu y.\varphi(y)), m\rangle, \top, \gamma))$.
- $\delta'(\langle p, \nu y.\varphi(y), m\rangle, \top, \gamma) = \mathsf{split}(\delta'(\langle p, \varphi(\nu y.\varphi(y)), m\rangle, \top, \gamma))$.

The definition of the function $\mathsf{split} : \mathcal{B}^+(D \times Q' \times \Gamma_\flat^*) \to \mathcal{B}^+(D \times Q' \times \Gamma_\flat^*)$ is a simple adaptation of the definition found in [KVW00]. For every $d \in D, q \in Q, m \in \{\exists, \forall\} \times \{p_e, p_s\}$ and $\beta \in \Gamma_\flat^*$ we have the following:

- $\mathsf{split}(\mathbf{true}) = \mathbf{true}, \mathsf{split}(\mathbf{false}) = \mathbf{false}$.
- $\mathsf{split}(\theta_1 \vee \theta_2) = \mathsf{split}(\theta_1) \vee \mathsf{split}(\theta_2)$ and $\mathsf{split}(\theta_1 \wedge \theta_2) = \mathsf{split}(\theta_1) \wedge \mathsf{split}(\theta_2)$.
- If $\psi \in cl(\varphi)$ is of the form $p, \neg p, [n]\psi', \langle n\rangle\psi', \mu y.\psi'(y)$ or $\nu y.\psi'(y)$, then $\mathsf{split}(d, \langle p, \psi, m\rangle, \beta) = (d, \langle p, \psi, m\rangle, \beta)$.
- $\mathsf{split}(d, \langle p, \psi_1 \vee \psi_2, m\rangle, \beta) = \mathsf{split}(d, \langle p, \psi_1, m\rangle, \beta) \vee \mathsf{split}(d, \langle p, \psi_2, m\rangle, \beta)$.
- $\mathsf{split}(d, \langle p, \psi_1 \wedge \psi_2, m\rangle, \beta) = \mathsf{split}(d, \langle p, \psi_1, m\rangle, \beta) \wedge \mathsf{split}(d, \langle p, \psi_2, m\rangle, \beta)$.

- It remains to define the acceptance condition F. Let d be the maximal alternation level of (fixpoint) sub-formulas of φ. Denote by G_i the set of all ν-formulas in $cl(\varphi)$ of alternation level i. Denote by B_i the set of all μ-formulas in $cl(\varphi)$ of alternation depth less than or equal to i. Now, $F = \{F_0, F_1, \ldots, F_{2d}\}$, where $F_0 = \emptyset$ and for every $1 \leq i \leq d$ we have $F_{2i-1} = F_{2i-2} \cup (Q \times B_i \times \{\forall, \exists\} \times \{p_e, p_s\})$, and $F_{2i} = F_{2i-1} \cup (Q \times G_i \times \{\forall, \exists\} \times \{p_e, p_s\})$. Clearly, $F_0 \subseteq F_1 \subseteq F_2 \subseteq \ldots \subseteq F_{2d}$. Since by the definition of PD-SPT a path π of a run r is accepting if the minimal i with $\mathsf{Inf}(\pi) \cap F_i \neq \emptyset$ is even, by our definition of F, such an index i corresponds to the outermost fixpoint formula that was visited infinitely often. Thus, the acceptance condition makes sure that, if a fixpoint formula is visited infinitely often, then this is a greatest fixpoint formula, and that all of its least fixpoint super-formulas are visited only finitely many times.

Let us now discuss the size of $\mathcal{A}_{\mathcal{S},\varphi}$. It is easy to see that $|Q'| = O(|Q| * |\varphi|)$, and $|\delta'| = O(|\delta| * |\varphi|)$. Hence, the size of $\mathcal{A}_{\mathcal{S},\varphi}$ is $O(|\mathcal{S}| * |\varphi|)$.

Finally, we show that $\mathcal{A}_{\mathcal{S},\varphi}$ is semi-alternating. It is sufficient to show that for every $(t,\beta) \in D$, $\sigma \in \Sigma$, $p,p' \in Q'$, and $\gamma \in \Gamma$, if $((t,\beta),p',\beta')$ appears in $\delta'(p,\sigma,\gamma)$ then $\beta = \beta'$. To see that, notice that $((t,\beta),p',\beta')$ appears in $\delta'(p,\sigma,\gamma)$ only if $vis(q,\beta') = (t,\beta)$, for some $q \in Q$. Since by definition (because the pushdown store is completely visible) we have that $vis(q,\beta') = (vis(q),\beta')$, and we are done. □

Theorem 2 implies that $\mathcal{M}_S \models_r \psi$ iff the language of the automaton $\mathcal{A}_{\mathcal{S},\neg\psi}$ is empty. We can now show the main result of the paper.

Theorem 3. *Given an* OPD $\mathcal{S}$ *and a formula* φ*, the pushdown module checking problem with imperfect state information is* 2EXPTIME*-complete if* φ *is a propositional or a graded* μ*-calculus formula, and* 3EXPTIME*-complete if* φ *is a* CTL* *formula.*

Proof. The lower bound follows from the known bound for pushdown module checking with perfect information (see [FMP07] for propositional and graded μ-calculus, and [BMP05] for *CTL**). For the upper bound, by Theorem 2, it is enough to check that $\mathcal{A}_{\mathcal{S},\neg\varphi}$ is empty. Recall that when considering propositional and graded μ-calculus (resp. *CTL**) $\mathcal{A}_{\mathcal{S},\neg\varphi}$ is a PD-SPT with $n = O(|\mathcal{S}| * |\varphi|)$ (resp. $n = O(|\mathcal{S}| * 2^{|\varphi|)})$ states, and index $k = O(|\varphi|)$. Let $\mathcal{M}_S = \langle AP, W_s, W_e, w_0, R, L, \cong\rangle$ be the module induced by $\mathcal{S}$. Observe that the set of directions of the strategy trees that are the input of $\mathcal{A}_{\mathcal{S},\neg\varphi}$ is $D = \{(vis(q),\beta) \mid ((p,\alpha),(q,\beta)) \in R \text{ for some } p,q,\alpha \text{ and } \beta\}$, and it is bounded from above by $|S|$. By Corollary 1, the emptiness of $\mathcal{A}_{\mathcal{S},\neg\varphi}$ can be decided in time double exponential in $|D|nk$. Thus, deciding if $\mathcal{M}_S \models_r \varphi$ can be done in time double-exponential in $|\mathcal{S}| * |\varphi|$ when considering propositional and graded μ-calculus, and triple-exponential in $|\mathcal{S}| * |\varphi|$ when considering *CTL**. □

Acknowledgment. The first and third author wish to thank Nir Piterman for useful discussions.

References

[AMV07] B. Aminof, A. Murano, and M.Y. Vardi. Pushdown module checking with imperfect information. In *CONCUR '07*, LNCS 4703, pages 461–476. Springer-Verlag, 2007.

[BLMV06] P.A. Bonatti, C. Lutz, A. Murano, and M.Y. Vardi. The complexity of enriched μ-calculi. In *ICALP'06*, LNCS 4052, pages 540–551, 2006.

[BMP05] Laura Bozzelli, Aniello Murano, and Adriano Peron. Pushdown module checking. In *LPAR'05*, LNCS 3835, pages 504–518. Springer-Verlag, 2005.

[CE81] 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.

[EH86] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *J. of the ACM*, 33(1):151–178, 1986.

[FM07] A. Ferrante and A. Murano. Enriched μ–calculus module checking. In *FOSSACS'07*, volume 4423 of *LNCS*, page 183197, 2007.

[FMP07] A. Ferrante, A. Murano, and M. Parente. Enriched μ–calculus pushdown module checking. In *LPAR'07*, volume 4790 of *LNAI*, pages 438–453, 2007.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 477–498. Springer-Verlag, 1985.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Koz83] D. Kozen. Results on the propositional mu–calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KPV02] O. Kupferman, N. Piterman, and M.Y. Vardi. Pushdown specifications. In *LPAR'02*, LNCS 2514, pages 262–277. Springer-Verlag, 2002.

[KSV02] O. Kupferman, U. Sattler, and M.Y. Vardi. The complexity of the graded μ-calculus. In *CADE'02*, LNAI 2392, pages 423–437, 2002.

[KV96] O. Kupferman and M.Y. Vardi. Module checking. In *CAV'96*, LNCS 1102, pages 75–86. Springer-Verlag, 1996.

[KV97] O. Kupferman and M. Y. Vardi. Module checking revisited. In *Proc. 9th International Computer Aided Verification Conference*, LNCS 1254, pages 36–47. Springer-Verlag, 1997.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. *J. of the ACM*, 47(2):312–360, 2000.

[KVW01] O. Kupferman, M.Y. Vardi, and P. Wolper. Module Checking. *Information and Computation*, 164(2):322–344, 2001.

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[QS81] 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.