Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)
Main Track

Module Checking of Pushdown Multi-agent Systems

Laura Bozzelli, **Aniello Murano**, **Adriano Peron**
University of Napoli “Federico II”
lr.bozzelli@gmail.com, nello.murano@gmail.com, adrperon@unina.it

Abstract

In this paper, we investigate the module-checking problem of pushdown multi-agent systems (PMS) against ATL and ATL* specifications. We establish that for ATL, module checking of PMS is 2EXPTIME-complete, which is the same complexity as pushdown module-checking for CTL. On the other hand, we show that ATL* module-checking of PMS turns out to be 4EXPTIME-complete, hence exponentially harder than both CTL* pushdown module-checking and ATL* model-checking of PMS. Our result for ATL* provides a rare example of a natural decision problem that is elementary yet but with a complexity that is higher than triply exponential-time.

1 Introduction

Model checking is a well-established formal-method technique to automatically check for global correctness of systems (Clarke and Emerson 1981; Queille and Sifakis 1981). Early use of model checking mainly considered *finite-state closed systems*, modelled as labelled state-transition graphs (Kripke structures) equipped with some internal degree of nondeterminism, and specifications given in terms of standard temporal logics such as the linear-time temporal logic LTL (Pnueli 1977) and the branching-time temporal logics CTL and CTL* (Emerson and Halpern 1986).

In the last two decades, model-checking techniques have been extended to the analysis of reactive and distributed component-based systems, where the behavior of a component depends on assumptions on its environment (the other components). One of the first approaches to model check *finite-state open systems* is *module checking* (Kupferman and Vardi 1996), a framework for handling the interaction between a system and an external unpredictable environment. In this setting, the states of the Kripke structure are partitioned into those controlled by the system and those controlled by the environment. The latter ones intrinsically carry an additional source of nondeterminism describing the possibility that the computation, from these states, can continue with any subset of its possible successor states. This means that while in model checking, we have only one computation tree representing the possible evolution of the system, in module checking we have an infinite number of trees to handle, one for each possible behavior of the environment. Deciding whether a system satisfies a property amounts to check that all such trees satisfy the property. This makes module checking harder to deal with. Classically, module checking has been investigated with respect to CTL and CTL* specifications. More recent approaches to the verification of multi-component finite-state systems (*multi-agent systems*) are based on the game paradigm: the system is modeled by a multi-player finite-state concurrent game, where at each step, the next state is determined by considering the “intersection” between the choices made simultaneously and independently by all the players. In this setting, properties are specified in logics for strategic reasoning such as the alternating-time temporal logics ATL and ATL* (Alur, Henzinger, and Kupferman 2002), well-known extensions of CTL and CTL*, respectively, which allow to express cooperation and competition among agents in order to achieve certain goals.

For a long time, there has been a common believe that module checking of CTL/CTL* is a special case of model checking of ATL/ATL*. The belief has been recently refuted in (Jamroga and Murano 2014). There, it was proved that module checking includes two features inherently absent in the semantics of ATL/ATL*, namely irrevocability and nondeterminism of strategies. On the other hand, temporal logics like CTL and CTL* do not accommodate strategic reasoning. These facts have motivated the extension of module checking to a finite-state multi-agent setting for handling specifications in ATL* (Jamroga and Murano 2015; Bozzelli and Murano 2017), which turns out to be more expressive than both CTL* module checking and ATL* model checking (Jamroga and Murano 2014; 2015).

Verification of pushdown systems. An active field of research is model checking of pushdown systems. These represent an infinite-state formalism suitable to capture the control flow of procedure calls and returns in programs. Model checking of (closed) pushdown systems against standard regular temporal logics (such as LTL, CTL, CTL*, and the modal μ-calculus) is decidable and it has been intensively studied in recent years leading to efficient verification algorithms and tools (see e.g. (Walukiewicz 1996; Bouajjani, Esparza, and Maler 1997; Ball and Rajamani 2000; Aminof, Kupferman, and Murano 2012; Aminof, Mogavero, and Murano 2014)). The verification of open pushdown systems in a two-player turn-based setting has been investigated in many works (see e.g. (Löding, Madhusudan, and Serre 2004; Hague and Ong 2009)). Open pushdown systems along with the module-checking paradigm have been considered in (Bozzelli, Murano, and Peron 2010). As in the case of finite-state systems, for the logic CTL (resp., CTL*), pushdown module-checking is singly exponentially harder than pushdown model-checking, being precisely 2EXPTIME-complete (resp., 3EXPTIME-complete), although with the same program complexity as pushdown model-checking (that is EXPTIME-complete). Pushdown module-checking has been investigated under several restrictions (Aminof et al. 2013; Bozzelli 2011; Murano, Napoli, and Parente 2008), including the imperfect-information setting case, where the latter variant is in general undecidable (Aminof et al. 2013). More recently in (Murano and Perelli 2015; Chen, Song, and Wu 2016), the verification of open pushdown systems has been extended to a concurrent game setting (*pushdown multi-agent systems*) by considering specifications in ATL* and the alternating-time modal μ-calculus.

Our contribution. In this paper, we extend the module-checking framework to the verification of multi-agent pushdown systems (PMS) by addressing the module-checking problem of PMS against ATL and ATL* specifications. We establish that ATL module-checking for PMS has the same complexity as pushdown module-checking for CTL, that is 2EXPTIME-complete. On the other hand, we show that ATL* module-checking of PMS has a very high complexity: it turns out to be exponentially harder than ATL* model-checking of PMS and pushdown module-checking for CTL*, being, precisely, 4EXPTIME-complete with an EXPTIME-complete complexity for a fixed-size formula. The upper bounds are obtained by an automata-theoretic approach. The matching lower bound for ATL* is shown by a technically non-trivial reduction from the word problem for 3EXPSPACE-bounded alternating Turing Machines. Our result for ATL* provides a rare example of a natural decision problem that is elementary yet but with a complexity that is higher than triply exponential-time. To the best of our knowledge, the unique known characterization of the class 4EXPTIME concerns validity of CTL* on alternating automata with bounded cooperative concurrency (Harel, Rosner, and Vardi 1990). Full proofs can be found in (Bozzelli, Murano, and Peron 2020).

2 Preliminaries

We fix the following notations. Let AP be a finite nonempty set of atomic propositions, Ag be a finite nonempty set of agents, and Ac be a finite nonempty set of actions that can be made by agents. For a set $A \subseteq Ag$ of agents, an A-decision d_A is an element in Ac^A assigning to each agent $a \in A$ an action $d_A(a)$. For $A, A' \subseteq Ag$ with $A \cap A' = \emptyset$, an A-decision d_A and A'-decision $d_{A'}$, $d_A \cup d_{A'}$ denotes the $(A \cup A')$-decision defined in the obvious way. Let $Dc = Ac^{Ag}$ be the set of *full decisions* of all the agents in Ag.

Let $\mathbb{N}$ be the set of natural numbers. For an infinite word w over an alphabet Σ and $i \geq 0$, $w(i)$ denotes the $(i+1)^{th}$ letter of w and $w_{\geq i}$ the suffix of w given by $w(i)w(i+1)\ldots$. For a finite word w over Σ, $|w|$ denotes the length of w.

Given a set Υ of directions, an (*infinite*) Υ-*tree* T is a prefix closed subset of Υ^* such that for all $\nu \in T$, $\nu \cdot \gamma \in T$ for some $\gamma \in \Upsilon$. Elements of T are called nodes and ε is the root of T. For $\nu \in T$, a child of ν in T is a node of the form $\nu \cdot \gamma$ for some $\gamma \in \Upsilon$. An (infinite) path of T is an infinite sequence π of nodes such that $\pi(i+1)$ is a child in T of $\pi(i)$ for all $i \geq 0$. For an alphabet Σ, a Σ-labeled Υ-tree is a pair $\langle T, Lab \rangle$ consisting of a Υ-tree and a labelling $Lab : T \mapsto \Sigma$ assigning to each node in T a symbol in Σ. We extend the labeling Lab to paths π in the obvious way, i.e. $Lab(\pi)$ is the infinite word over Σ given by $Lab(\pi(0))Lab(\pi(1))\ldots$. The labeled tree $\langle T, Lab \rangle$ is *complete* if $T = \Upsilon^*$. Given $k \in \mathbb{N} \setminus \{0\}$, a *k-ary tree* is a $\{1, \ldots, k\}$-tree.

Concurrent game structures (CGS) CGS (Alur, Henzinger, and Kupferman 2002) extend Kripke structures to a setting with multiple agents. They can be viewed as multi-player games in which players perform concurrent actions, chosen strategically as a function of the history of the game.

Definition 2.1 (CGS). A CGS (over AP, Ag, and Ac) is a tuple $\mathcal{G} = \langle S, s_0, Lab, \tau \rangle$, where S is a countable set of states, $s_0 \in S$ is the initial state, $Lab : S \mapsto 2^{AP}$ maps each state to a set of atomic propositions, and $\tau : S \times Dc \mapsto S \cup \{\dashv\}$ is a transition function that maps a state and a full decision either to a state or to the special symbol $\dashv$ ($\dashv$ is for 'undefined') such that for all states s, there exists $d \in Dc$ so that $\tau(s, d) \neq \dashv$. Given a set $A \subseteq Ag$ of agents, an A-decision d_A, and a state s, we say that d_A *is available at state* s if there exists an $(Ag \setminus A)$-decision $d_{Ag \setminus A}$ such that $\tau(s, d_A \cup d_{Ag \setminus A}) \in S$.

For a state s and an agent a, *state* s *is controlled by* a if there is a unique $(Ag \setminus \{a\})$-decision available at state s. Agent a *is passive in* s if there is a unique $\{a\}$-decision available at state s. A *multi-agent turn-based game* is a CGS where each state is controlled by an agent.

Note that in modelling independent agents, usually one assume that at each state s, each agent a has a set $Ac_{a,s} \subseteq Ac$ of actions which are enabled at state s. This is reflected in the transition function τ by requiring that the set of full decisions d such that $\tau(s, d) \neq \dashv$ corresponds to $(Ac_{a,s})_{a \in Ag}$. We now recall the notion of strategy in a CGS $\mathcal{G} = \langle S, s_0, Lab, \tau \rangle$. A *play* is an infinite sequence of states $s_1 s_2 \ldots$ such that for all $i \geq 1$, s_{i+1} is a *successor* of s_i, i.e. $s_{i+1} = \tau(s_i, d)$ for some full decision d. A *track* (or *history*) ν is a nonempty prefix of some play. Given a set $A \subseteq Ag$ of agents, a *strategy for* A is a mapping f_A assigning to each track ν (representing the history the agents saw so far) an A-decision available at the last state, denoted $lst(\nu)$, of ν. The *outcome* function $out(s, f_A)$ for a state s and the strategy f_A returns the set of all the plays starting at state s that can occur when agents A execute strategy f_A from state s on. Formally, $out(s, f_A)$ is the set of plays $\pi = s_1 s_2 \ldots$ such that $s_1 = s$ and for all $i \geq 1$, there is $d \in Ac^{Ag \setminus A}$ so that $s_{i+1} = \tau(s_i, f_A(s_1 \ldots s_i) \cup d)$.

Definition 2.2. For a set Υ of directions, a *Concurrent Game* Υ-*Tree* (Υ-CGT) is a CGS $\langle T, \varepsilon, Lab, \tau \rangle$, where $\langle T, Lab \rangle$ is a 2^{AP}-labeled Υ-tree, and for each node $x \in T$, the successors of x correspond to the children of x in T. Every CGS $\mathcal{G} = \langle S, s_0, Lab, \tau \rangle$ induces a S-CGT $Unw(\mathcal{G})$ obtained by unwinding $\mathcal{G}$ from the initial state. Formally, $Unw(\mathcal{G}) = \langle T, \varepsilon, Lab', \tau' \rangle$, where $\nu \in T$ iff $s_0 \cdot \nu$ is a track of $\mathcal{G}$, and for all $\nu \in T$ and $d \in Dc$, $Lab'(\nu) = Lab(lst(\nu))$ and $\tau'(\nu, d) = \nu \cdot \tau(lst(\nu), d)$, where $lst(\varepsilon) = s_0$.

Pushdown multi-agent systems (PMS) PMS, introduced in (Murano and Perelli 2015), generalize standard pushdown systems to a concurrent multi-player setting.

Definition 2.3. A PMS (over AP, Ag, and Ac) is a tuple $\mathcal{S} = \langle Q, \Gamma \cup \{\gamma_0\}, q_0, Lab, \Delta\rangle$, where Q is a finite set of (control) states, $\Gamma \cup \{\gamma_0\}$ is a finite stack alphabet (γ_0 is the special *stack bottom symbol*), $q_0 \in Q$ is the initial state, $Lab : Q \mapsto 2^{AP}$ maps each state to a set of atomic propositions, and $\Delta : Q \times (\Gamma \cup \{\gamma_0\}) \times Dc \mapsto (Q \times \Gamma^*) \cup \{\dashv\}$ is a transition function ($\dashv$ is for 'undefined') s.t. for all $(q,\gamma) \in Q \times (\Gamma \cup \{\gamma_0\})$, there is $d \in Dc$ so that $\Delta(q,\gamma,d) \neq \dashv$.

The size $|\Delta|$ of the transition function Δ is given by $|\Delta| = \sum_{(q',\beta)\in\Delta(q,\gamma,d)} |\beta|$. A *configuration* of the PMS $\mathcal{S}$ is a pair (q,β) where q is a (control) state and $\beta \in \Gamma^* \cdot \gamma_0$ is a stack content. Intuitively, when the PMS $\mathcal{S}$ is in state q, the stack top symbol is γ and the agents take a full decision d available at the current configuration, i.e. such that $\Delta(q,\gamma,d) = (q',\beta)$ for some $(q',\beta) \in Q \times \Gamma^*$, then $\mathcal{S}$ moves to the configuration with state q' and stack content obtained by removing γ and pushing β (if $\gamma = \gamma_0$ then γ is not removed). Formally, the PMS $\mathcal{S}$ induces the infinite-state CGS $\mathcal{G}(\mathcal{S}) = \langle S, s_0, Lab', \tau\rangle$, where S is the set of configurations of $\mathcal{S}$, $s_0 = (q_0, \gamma_0)$ (initially, the stack contains just the bottom symbol γ_0), $Lab'((q,\beta)) = Lab(q)$ for each configuration (q,β), and the transition function τ is defined as follows for all $((q,\gamma\cdot\beta),d) \in S \times Dc$, where $\gamma \in \Gamma \cup \{\gamma_0\}$:

- *either* $\Delta(q,\gamma,d) = \dashv$ and $\tau((q,\gamma\cdot\beta),d) = \dashv$,
- *or* $\gamma \in \Gamma$, $\Delta(q,\gamma,d) = (q',\beta')$, and $\tau((q,\gamma\cdot\beta),d) = (q',\beta'\cdot\beta)$,
- *or* $\gamma = \gamma_0$ (hence, $\beta = \varepsilon$), $\Delta(q,\gamma,d) = (q',\beta')$, and $\tau((q,\gamma\cdot\beta),d) = (q',\beta'\cdot\gamma_0)$.

2.1 The Logics ATL* and ATL

We recall the alternating-temporal logics ATL* and ATL proposed by Alur et al. (Alur, Henzinger, and Kupferman 2002) as extensions of the standard branching-time temporal logics CTL* and CTL (Emerson and Halpern 1986), where the path quantifiers are replaced by more general parameterized quantifiers which allow for reasoning about the strategic capability of groups of agents. For the given sets AP and Ag of atomic propositions and agents, ATL* formulas φ are defined as:

$$\varphi ::= \texttt{true} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi\,\mathsf{U}\,\varphi \mid \langle\langle A\rangle\rangle\varphi$$

where $p \in AP$, $A \subseteq Ag$, X and U are the standard "next" and "until" temporal modalities, and $\langle\langle A\rangle\rangle$ is the "existential strategic quantifier" parameterized by a set of agents. Formula $\langle\langle A\rangle\rangle\varphi$ expresses that the group of agents A has a collective strategy to enforce property φ. We use some shorthand: $\mathsf{F}\varphi := \texttt{true}\,\mathsf{U}\,\varphi$ ("eventually") and $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$ ("always"). A *state formula* is a formula where each temporal modality is in the scope of a strategic quantifier. A *basic formula* is a state formula of the form $\langle\langle A\rangle\rangle\varphi$. The logic ATL is the fragment of ATL* where each temporal modality is immediately preceded by a strategic quantifier. Note that CTL* (resp., CTL) corresponds to the fragment of ATL* (resp., ATL), where only the strategic modalities $\langle\langle Ag\rangle\rangle$ and $\langle\langle\emptyset\rangle\rangle$ (equivalent to the existential and universal path quantifiers E and A, respectively) are allowed.

Given a CGS $\mathcal{G}$ with labeling Lab and a play π of $\mathcal{G}$, the satisfaction relation $\mathcal{G},\pi \models \varphi$ for ATL* is defined as follows (Boolean connectives are treated as usual):

$$\begin{array}{ll}
\mathcal{G},\pi \models p & \Leftrightarrow p \in Lab(\pi(0)) \\
\mathcal{G},\pi \models \mathsf{X}\varphi & \Leftrightarrow \mathcal{G},\pi_{\geq 1} \models \varphi \\
\mathcal{G},\pi \models \varphi_1\,\mathsf{U}\,\varphi_2 & \Leftrightarrow \text{ there is } j \geq 0 : \mathcal{G},\pi_{\geq j} \models \varphi_2 \text{ and} \\
& \quad \mathcal{G},\pi_{\geq k} \models \varphi_1 \text{ for all } 0 \leq k < j \\
\mathcal{G},\pi \models \langle\langle A\rangle\rangle\varphi & \Leftrightarrow \text{for some strategy } f_A \text{ for } A, \\
& \quad \mathcal{G},\pi' \models \varphi \text{ for all } \pi' \in out(\pi(0), f_A)
\end{array}$$

For a state s of $\mathcal{G}$, $\mathcal{G}, s \models \varphi$ if there is a play π starting from s such that $\mathcal{G},\pi \models \varphi$. Note that if φ is a state formula, then for all plays π and π' from s, $\mathcal{G},\pi \models \varphi$ iff $\mathcal{G},\pi' \models \varphi$. $\mathcal{G}$ is a model of φ, denoted $\mathcal{G} \models \varphi$, if for the initial state s_0, $\mathcal{G}, s_0 \models \varphi$. Note that $\mathcal{G} \models \varphi$ iff $Unw(\mathcal{G}) \models \varphi$.

2.2 ATL* and ATL Pushdown Module-checking

In this section, we first recall the ATL* module-checking framework which turns out to be more expressive than both CTL* module-checking and ATL* model-checking (Jamroga and Murano 2014; 2015). Then, we generalize this setting to pushdown multi-agent systems.

In the multi-agent module-checking setting, one consider CGS with a distinguished agent (the *environment*).

Definition 2.4 (Open CGS)**.** An open CGS is a CGS $\mathcal{G} = \langle S, s_0, Lab, \tau\rangle$ containing a special agent called "the environment" ($env \in Ag$). Moreover, for every state s, either s is controlled by the environment (environment state) or the environment is passive in s (system state).

For an open CGS $\mathcal{G} = \langle S, s_0, Lab, \tau\rangle$, the set of *environment strategy trees of* $\mathcal{G}$, denoted $exec(\mathcal{G})$, is the set of S-CGT obtained from $Unw(\mathcal{G})$ by possibly pruning some environment transitions. Formally, $exec(\mathcal{G})$ is the set of S-CGT $\mathcal{T} = \langle T, \varepsilon, Lab', \tau'\rangle$ such that T is a prefix closed subset of the set of $Unw(\mathcal{G})$-nodes and for all $\nu \in T$ and $d \in Dc$, $Lab'(\nu) = Lab(lst(\nu))$, and $\tau'(\nu,d) = \nu \cdot \tau(lst(\nu),d)$ if $\nu \cdot \tau(lst(\nu),d) \in T$, and $\tau'(\nu,d) = \dashv$ otherwise, where $lst(\varepsilon) = s_0$. Moreover, for all $\nu \in T$, the following holds:

- if $lst(\nu)$ is a system state, then for each successor s of $lst(\nu)$ in $\mathcal{G}$, $\nu \cdot s \in T$;
- if $lst(\nu)$ is an environment state, then there is a nonempty subset $\{s_1,\ldots,s_n\}$ of the set of $lst(\nu)$-successors such that the set of children of ν in T is $\{\nu\cdot s_1,\ldots,\nu\cdot s_n\}$.

Intuitively, when $\mathcal{G}$ is in a system state s, then all the transitions from s are enabled. When $\mathcal{G}$ is instead in an environment state, the set of enabled transitions from s depend on the current environment. Since the behavior of the environment is nondeterministic, we have to consider all the possible subsets of the set of s-successors. The only constraint, since we consider environments that cannot block the system, is that not all the transitions from s can be disabled. For an open CGS $\mathcal{G}$ and an ATL* formula φ, $\mathcal{G}$ *reactively satisfies* φ, denoted $\mathcal{G} \models^r \varphi$, if for all strategy trees $\mathcal{T} \in exec(\mathcal{G})$, $\mathcal{T} \models \varphi$. Note that $\mathcal{G} \models^r \varphi$ implies $\mathcal{G} \models \varphi$ (since $Unw(\mathcal{G}) \in exec(\mathcal{G})$), but the converse in general does not hold.

ATL* and ATL Pushdown Module-checking. An *open PMS* is a PMS $\mathcal{S}$ such that the induced CGS $\mathcal{G}(\mathcal{S})$ is open.

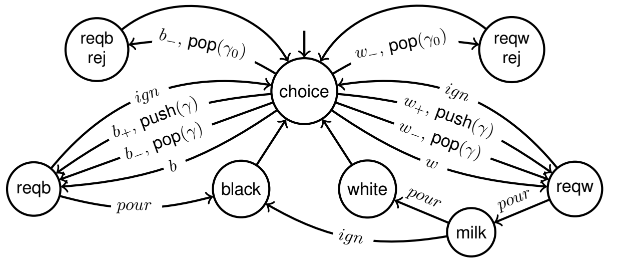

Figure 1: Multi-agent pushdown coffee machine $\mathcal{S}_{cof}$

Note that for an open PMS, the property of a configuration of being an environment or system configuration depends only on the control state and the symbol on the top of the stack. The *pushdown module-checking problem against* ATL (resp., ATL*) is checking for a given open PMS $\mathcal{S}$ and an ATL formula (resp., ATL* state formula) φ whether $\mathcal{G}(\mathcal{S}) \models^r \varphi$.

Example 2.5. Consider a coffee machine that allows customers (representing the environment) to choose between the following actions: (i) ordering and paying a black or white coffee (actions b or w), (ii) like the actions in the previous point but, additionally, paying a "suspended" coffee for the benefit of any customer unknown (actions b_+ or w_+), and (iii) asking for a gifted (black or white) coffee (actions b_- or w_-). The coffee machine is modeled by a turn-based open PMS $\mathcal{S}_{cof}$ with three agents: the environment, the brewer br whose function is to pour coffee into the cup (action $pour$), and the milk provider who can add milk (action $milk$). The two system agents can be faulty and ignore the request from the environment (action ign). The stack is exploited for keeping track of the number of suspended coffee: a request for a gifted coffee can be accepted only if the stack is not empty. After the completion of a request, the machine waits for further selections. The PMS $\mathcal{S}_{cof}$ is represented as a graph in Figure 1 where each node (control state) is labeled by the propositions holding at it: the state labeled by choice is controlled by the environment, the states labeled by reqb or reqw are controlled by the brewer br, while the state labeled by milk is controlled by the milk provider. The notation push(γ) denotes a push stack operation (pushing the symbol $\gamma \neq \gamma_0$), while pop(γ) (resp., pop(γ_0)) denotes a pop operation onto a non-empty (resp., empty) stack.

In module checking, we can condition the property to be achieved on the behaviour of the environment. For instance, users who never order white coffee and whose request is never rejected can be served by the brewer alone: $\mathcal{G}(\mathcal{S}_{cof}) \models^r \mathsf{AG}(\neg reqw \wedge \neg rej) \rightarrow \langle\langle br \rangle\rangle \mathsf{F}\,\mathsf{black}$. In model checking, the same formula does not express any interesting property since $\mathcal{G}(\mathcal{S}_{cof}) \not\models \mathsf{AG}(\neg reqw \wedge \neg rej)$. Likewise $\mathcal{G}(\mathcal{S}_{cof}) \models \mathsf{AG}\neg reqw \rightarrow \langle\langle br \rangle\rangle \mathsf{F}\,\mathsf{black}$, whereas module checking gives a different and more intuitive answer: $\mathcal{G}(\mathcal{S}_{cof}) \not\models^r \mathsf{AG}\neg reqw \rightarrow \langle\langle br \rangle\rangle \mathsf{F}\,\mathsf{black}$ (there are environments where requests for a gifted coffee are always rejected).

3 Decision Procedures

In this section, we provide an automata-theoretic framework for solving the pushdown module-checking problem against ATL and ATL* which is based on the use of alternating automata for CGS (ACG) (Schewe and Finkbeiner 2006) and nondeterministic pushdown tree automata (NPTA) (Kupferman, Piterman, and Vardi 2002). For the given open PMS $\mathcal{S}$ and ATL formula (resp., ATL* state formula) φ, by exploiting known results, we first build in linear-time (resp., double exponential time) a parity ACG $\mathcal{A}_{\neg\varphi}$ accepting the set of CGT which satisfy $\neg\varphi$. Then, as a main result, we show how to construct a parity NPTA $\mathcal{P}$ accepting suitable encodings of the strategy trees of $\mathcal{G}(\mathcal{S})$ accepted by $\mathcal{A}_{\neg\varphi}$. Hence, $\mathcal{G}(\mathcal{S}) \models_r \varphi$ iff the language accepted by $\mathcal{P}$ is empty.

In the following, we first recall the frameworks of NPTA and ACG, and known translations of ATL* and ATL formulas into equivalent parity ACG. Then, in Subsection 3.1, by exploiting parity NPTA, we show that given an open PMS $\mathcal{S}$ and a parity ACG $\mathcal{A}$, checking that no strategy tree of $\mathcal{G}(\mathcal{S})$ is accepted by $\mathcal{A}$ can be done in time double exponential in the size of $\mathcal{A}$ and singly exponential in the size of $\mathcal{S}$.

Nondeterministic Pushdown Tree Automata (NPTA). We describe parity NPTA over labeled complete k-ary trees for a given $k \geq 1$, which are tuples $\mathcal{P} = \langle \Sigma, Q, \Gamma \cup \{\gamma_0\}, q_0, \rho, \Omega \rangle$, where Σ is a finite input alphabet, Q, $\Gamma \cup \{\gamma_0\}$, and q_0 are defined as for PMS, $\rho : Q \times \Sigma \times (\Gamma \cup \{\gamma_0\}) \rightarrow 2^{(Q \times \Gamma^*)^k}$ is a transition function, and $\Omega : Q \mapsto \mathbb{N}$ is a parity acceptance condition over Q assigning to each state a color. The *index* of $\mathcal{P}$ is the number of colors in Ω, i.e., the cardinality of $\Omega(Q)$. A run of the NPTA $\mathcal{P}$ on a Σ-labeled complete k-ary tree $\langle T, \mathit{Lab} \rangle$ (with $T = \{1, \ldots, k\}^*$) is a $(Q \times \Gamma^*.\gamma_0)$-labeled tree $r = \langle T, \mathit{Lab}_r \rangle$ such that $\mathit{Lab}_r(\varepsilon) = (q_0, \gamma_0)$ (initially, the stack is empty) and for each $x \in T$ with $\mathit{Lab}_r(x) = (q, \gamma \cdot \beta)$, there is $\langle (q_1, \beta_1), \ldots, (q_k, \beta_k) \rangle \in \rho(q, \mathit{Lab}(x), \gamma)$ such that for all $1 \leq i \leq k$, $\mathit{Lab}_r(x \cdot i) = (q_i, \beta_i \cdot \beta)$ if $\gamma \neq \gamma_0$, and $\mathit{Lab}_r(x \cdot i) = (q_i, \beta_i \cdot \gamma_0)$ otherwise (note that in this case $\beta = \varepsilon$). The run $r = \langle T, \mathit{Lab}_r \rangle$ is accepting if for all infinite paths π starting from the root, the highest color $\Omega(q)$ of the states q appearing infinitely often along $\mathit{Lab}_r(\pi)$ is even. The language $\mathcal{L}(\mathcal{P})$ accepted by $\mathcal{P}$ consists of the Σ-labeled complete k-ary trees $\langle T, \mathit{Lab} \rangle$ such that there is an accepting run of $\mathcal{P}$ over $\langle T, \mathit{Lab} \rangle$.

For complexity analysis, we consider two parameters: the size $|\rho|$ of ρ given by $\sum_{\langle (q_1,\beta_1),\ldots,(q_k,\beta_k) \rangle \in \rho(q,\sigma,\gamma)} |\beta_1| + \ldots + |\beta_k|$ and the smaller parameter $||\rho||$ given by $||\rho|| = \sum_{\beta \in \rho_0} |\beta|$ where ρ_0 is the set of words $\beta \in \Gamma^*.\gamma_0$ occurring in ρ. The following result has been established in (Kupferman, Piterman, and Vardi 2002) (see also (Bozzelli, Murano, and Peron 2010)).

Proposition 3.1. *(Kupferman, Piterman, and Vardi 2002; Bozzelli, Murano, and Peron 2010) The emptiness problem for a parity NPTA of index m with n states, and transition function ρ can be solved in time $O(|\rho| \cdot 2^{O(||\rho||^2 \cdot n^2 \cdot m^2 \log m)})$.*

Alternating automata for CGS (ACG for short) (Schewe and Finkbeiner 2006). ACG generalize alternating automata by branching universally or existentially over all successors that result from the agents' decisions. Formally, for a set X, let $\mathbb{B}^+(X)$ be the set of *positive* Boolean formulas over X, i.e. Boolean formulas built from elements in X using $\vee$ and $\wedge$. A subset Y of X is a *model* of $\theta \in \mathbb{B}^+(X)$

if the truth assignment that assigns true (resp., false) to the elements in Y (resp., $X \setminus Y$) satisfies θ. A parity ACG over 2^{AP} and Ag is a tuple $\mathcal{A} = \langle Q, q_0, \delta, \Omega \rangle$, where Q, q_0, and Ω are defined as for NPTA, while δ is a transition function of the form $\delta : Q \times 2^{AP} \to \mathbb{B}^+(Q \times \{\Box, \Diamond\} \times 2^{Ag})$. The transition function δ maps a state and an input letter to a positive Boolean combination of universal atoms $(q, \Box, A)$ which refer to *all* successors states for some available A-decision, and existential atoms $(q, \Diamond, A)$ which refer to *some* successor state for all available A-decisions. The size $|\mathcal{A}|$ of $\mathcal{A}$ is $|Q| + |\mathit{Atoms}(\mathcal{A})|$, where $\mathit{Atoms}(\mathcal{A})$ is the set of atoms of $\mathcal{A}$, i.e. the set of tuples in $Q \times \{\Box, \Diamond\} \times 2^{Ag}$ occurring in the transition function δ.

We interpret the parity ACG $\mathcal{A}$ over CGT. Given a CGT $\mathcal{T} = \langle T, \varepsilon, \mathit{Lab}, \tau \rangle$ over AP and Ag, a run of $\mathcal{A}$ over $\mathcal{T}$ is a $(Q \times T)$-labeled $\mathbb{N}$-tree $r = \langle T_r, \mathit{Lab}_r \rangle$, where each node of T_r labelled by (q, ν) describes a copy of the automaton that is in the state q and reads the node ν of T. Moreover, we require that $r(\varepsilon) = (q_0, \varepsilon)$ (initially, the automaton is in state q_0 reading the root node), and for each $y \in T_r$ with $r(y) = (q, \nu)$, there is a set $H \subseteq Q \times \{\Box, \Diamond\} \times 2^{Ag}$ such that H is a model of $\delta(q, \mathit{Lab}(\nu))$ and the set L of labels associated with the children of y in T_r satisfies the following:

- for all universal atoms $(q', \Box, A) \in H$, there is an available A-decision d_A in the node ν of $\mathcal{T}$ such that for all the children ν' of ν which are consistent with d_A, $(q', \nu') \in L$;
- for all existential atoms $(q', \Diamond, A) \in H$ and for all available A-decisions d_A in the node ν of $\mathcal{T}$, there is some child ν' of ν which is consistent with d_A such that $(q', \nu') \in L$.

The run r is accepting if for all infinite paths π starting from the root, the highest color of the states appearing infinitely often along $\mathit{Lab}_r(\pi)$ is even. The language $\mathcal{L}(\mathcal{A})$ accepted by $\mathcal{A}$ consists of the CGT $\mathcal{T}$ over AP and Ag such that there is an accepting run of $\mathcal{A}$ over $\mathcal{T}$.

We exploit a known translation of ATL* state formulas (resp., ATL formulas) into equivalent parity ACG which has been provided in (Bozzelli and Murano 2017). In particular, the following holds, where for a finite set B disjunct from AP and a CGT $\mathcal{T} = \langle T, \varepsilon, \mathit{Lab}, \tau \rangle$ over AP, a *B-labeling extension of $\mathcal{T}$* is a CGT over $AP \cup B$ of the form $\langle T, \varepsilon, \mathit{Lab}', \tau \rangle$, where $\mathit{Lab}'(\nu) \cap AP = \mathit{Lab}(\nu)$ for all $\nu \in T$.

Theorem 3.2. *(Bozzelli and Murano 2017) For an ATL* state formula (resp., ATL formula) φ over AP, one can construct in doubly exponential time (resp., linear time) a parity ACG $\mathcal{A}_\varphi$ over $2^{AP \cup B_\varphi}$, where B_φ is the set of basic subformulas of φ, such that for all CGT $\mathcal{T}$ over AP, $\mathcal{T}$ is a model of φ iff there exists a B_φ-labeling extension of $\mathcal{T}$ which is accepted by $\mathcal{A}_\varphi$. Moreover, $\mathcal{A}_\varphi$ has size $O(2^{2^{O(|\Phi| \cdot \log(|\varphi|))}})$ and index $2^{O(|\varphi|)}$ (resp., size $O(|\varphi|)$ and index 2).*

3.1 Upper Bounds for ATL and ATL* Pushdown Module-checking

Let $\mathcal{S}$ be an open PMS, φ an ATL* (resp., ATL) formula, and $\mathcal{A}_{\neg\varphi}$ the parity ACG over $2^{AP \cup B_\varphi}$ (B_φ is the set of basic subformulas of φ) of Theorem 3.2 associated with the negation of φ. By Theorem 3.2, checking that $\mathcal{G}(\mathcal{S}) \models_r \varphi$ reduces to check that there are no B_φ-labeling extensions of the strategy trees of $\mathcal{G}(\mathcal{S})$ accepted by $\mathcal{A}_{\neg\varphi}$. In this section, we provide an algorithm for checking this last condition. In particular, we establish the following result.

Theorem 3.3. *Given an open PMS $\mathcal{S}$ on AP, a finite set B disjunct from AP, and a parity ACG $\mathcal{A}$ on $2^{AP \cup B}$, checking that there are no B-labeling extensions of strategy trees of $\mathcal{G}(\mathcal{S})$ accepted by $\mathcal{A}$ can be done in time doubly exponential in the size of $\mathcal{A}$ and singly exponential in the size of $\mathcal{S}$.*

By Theorems 3.2 and 3.3, and since pushdown module-checking against CTL is already 2EXPTIME-complete, and EXPTIME-complete for a fixed CTL formula (Bozzelli, Murano, and Peron 2010), we obtain the following corollary.

Corollary 3.4. *Pushdown module-checking for ATL* is in* 4EXPTIME *while pushdown module-checking for ATL is* 2EXPTIME*-complete. Moreover, for a fixed ATL* state formula (resp., ATL formula), the pushdown module-checking problem is* EXPTIME*-complete.*

In Section 4, we provide a lower bound for ATL* matching the upper bound in the corollary above. We now illustrate the proof of Theorem 3.3 which is based on a reduction to emptiness of parity NPTA. For simplicity, we assume that the set B in the statement of Theorem 3.3 is empty (the general case where $B \neq \emptyset$ is similar).

Fix an open PMS $\mathcal{S} = \langle Q, \Gamma \cup \{\gamma_0\}, q_0, \mathit{Lab}, \Delta \rangle$ on AP and a parity ACG $\mathcal{A}$ on 2^{AP}, and let $\mathcal{G}(\mathcal{S}) = \langle S, s_0, \mathit{Lab}_S, \tau \rangle$. For all pairs $(q, \gamma) \in Q \times (\Gamma \cup \{\gamma_0\})$, let $\mathit{next}_\mathcal{S}(q, \gamma)$ be the finite set of pairs $(q', \beta) \in Q \times \Gamma^*$ s.t. there is a full decision d so that $\Delta(q, \gamma, d) = (q', \beta)$. We fix an ordering on the set $\mathit{next}_\mathcal{S}(q, \gamma)$ which induces an ordering on the finite set of successors of all the configurations of the form $(q, \gamma \cdot \alpha)$, and we consider the parameter $k_\mathcal{S} = \max\{|\mathit{next}_\mathcal{S}(q, \gamma)| \mid (q, \gamma) \in Q \times (\Gamma \cup \{\gamma_0\})\}$ representing the finite branching degree of $\mathit{Unw}(\mathcal{G}(\mathcal{S}))$. Thus, we can encode each track $\nu = s_0, s_1, \ldots, s_n$ of $\mathcal{G}(\mathcal{S})$ starting from the initial state, by the finite word $i_1, \ldots, i_n$ over $\{1, \ldots, k_\mathcal{S}\}$ of length n where for all $1 \leq h \leq n$, i_h represents the index of state s_h in the ordered set of successors of state s_{h-1}. Now, we observe that the transition function τ' of a strategy tree $\mathcal{T} = \langle T, \varepsilon, \mathit{Lab}', \tau' \rangle$ of $\mathcal{G}(\mathcal{S})$ is completely determined by T and the transition function τ of $\mathcal{G}(\mathcal{S})$. Hence, for the fixed open CGS $\mathcal{G}(\mathcal{S})$, $\mathcal{T}$ can be simply specified by the underlying 2^{AP}-labeled tree $\langle T, \mathit{Lab}' \rangle$. We consider an equivalent representation of $\langle T, \mathit{Lab}' \rangle$ by a $(2^{AP} \cup \{\bot\})$-labeled *complete* $k_\mathcal{S}$-tree $\langle \{1, \ldots, k_\mathcal{S}\}^*, \mathit{Lab}_\bot \rangle$, called the $\bot$-completion encoding of $\mathcal{T}$ ($\bot$ is a fresh proposition), where the labeling $\mathit{Lab}_\bot$ is defined as follows for each node $x \in \{1, \ldots, k_\mathcal{S}\}^*$:

- if x encodes a track $s_0 \cdot \nu$ such that ν is a node of T, then $\mathit{Lab}_\bot(x) = \mathit{Lab}'(\nu)$ (*concrete nodes*);
- otherwise, $\mathit{Lab}(x) = \{\bot\}$ (*completion nodes*).

In this way, all the labeled trees encoding strategy trees $\mathcal{T}$ of $\mathcal{G}(\mathcal{S})$ have the same structure (they all coincide with $\{1, \ldots, k_\mathcal{S}\}^*$), and they differ only in their labeling. Thus, the proposition $\bot$ is used to denote both "completion" nodes and nodes in $\mathit{Unw}(\mathcal{G}(\mathcal{S}))$ which are absent in $\mathcal{T}$ (possible environment choices are disabled). We show the following result which, together with Proposition 3.1, provides a proof of Theorem 3.3 (for the case $B = \emptyset$).

Theorem 3.5. *For an open* PMS $\mathcal{S} = \langle Q, \Gamma \cup \{\gamma_0\}, q_0, Lab, \Delta \rangle$ *over* AP *and a parity* ACG $\mathcal{A} = \langle Q_\mathcal{A}, q^0_\mathcal{A}, \delta, \Omega \rangle$ *over* 2^{AP} *with index* h, *one can construct in single exponential time, a parity* NPTA $\mathcal{P}$ *accepting the set of the* $\perp$*-completion encodings of the strategy trees of* $\mathcal{G}(\mathcal{S})$ *which are accepted by* $\mathcal{A}$. *Moreover,* $\mathcal{P}$ *has index* $O(h|\mathcal{A}|^2)$, *number of states* $O(|Q| \cdot (h|\mathcal{A}|^2)^{O(h|\mathcal{A}|^2)})$, *and transition function* ρ *such that* $||\rho|| = O(|\Delta| \cdot (h|\mathcal{A}|^2)^{O(h|\mathcal{A}|^2)})$.

Sketched Proof. First, we observe that for the given parity ACG $\mathcal{A}$ and an input CGT $\mathcal{T}$, we can associate in a standard way to $\mathcal{A}$ and $\mathcal{T}$ an infinite-state parity game, where player 0 plays for acceptance, while player 1 plays for rejection. Winning strategies of player 0 correspond to accepting runs of $\mathcal{A}$ over $\mathcal{T}$. Thus, since the existence of a winning strategy in parity games implies the existence of a memoryless one, we can restrict ourselves to consider only memoryless runs of $\mathcal{A}$, i.e. runs $r = \langle T_r, Lab_r \rangle$ where the behavior of $\mathcal{A}$ along r depends only on the current input node and current state. Formally, r is memoryless if for all nodes y and y' of r having the same label, the subtrees rooted at the nodes y and y' of r are isomorphic. We now provide a representation of the memoryless runs of $\mathcal{A}$ over the strategy trees of the open CGS $\mathcal{G}(\mathcal{S})$ induced by the given open PMS $\mathcal{S}$.

Fix a strategy tree $\mathcal{T} = \langle T, \varepsilon, Lab_\mathcal{T}, \tau \rangle$ of $\mathcal{G}(\mathcal{S})$ and let $\langle \{1, \ldots, k_\mathcal{S}\}^*, Lab_\perp \rangle$ be the $\perp$-completion encoding of $\mathcal{T}$. Recall that $Atoms(\mathcal{A})$ is the set of atoms of $\mathcal{A}$, i.e. the set of tuples in $Q_\mathcal{A} \times \{\Box, \Diamond\} \times 2^{Ag}$ occurring in the transition function δ of $\mathcal{A}$. Let $Ann = 2^{Q_\mathcal{A} \times Atoms(\mathcal{A})}$ be the finite set of *annotations* and $\Upsilon = (2^{AP} \times Ann \times Ann) \cup \{\perp\}$. For an annotation $an \in Ann$, we denote by $Dom(an)$ the set of $\mathcal{A}$-states q such that $(q, atom) \in an$ for some atom $atom \in Atoms(\mathcal{A})$, and by $Cod(an)$ the set of states occurring in the atoms of an. We represent memoryless runs r of $\mathcal{A}$ over $\mathcal{T}$ as *annotated extensions* of the $\perp$-completion encoding $\langle \{1, \ldots, k_\mathcal{S}\}^*, Lab_\perp \rangle$ of $\mathcal{T}$, i.e. Υ-labeled complete $k_\mathcal{S}$-trees $\langle \{1, \ldots, k_\mathcal{S}\}^*, Lab_\Upsilon \rangle$, where for every concrete node $x \in \{1, \ldots, k_\mathcal{S}\}$ encoding a node ν_x of T, $Lab_\Upsilon(x)$ is of the form $(Lab_\perp(x), an, an')$ (recall that $Lab_\perp(x) = Lab_\mathcal{T}(\nu_x)$), and for every completion node x, $Lab_\Upsilon(x) = Lab_\perp(x) = \{\perp\}$. Intuitively, the meaning of the first annotation an in the label of a concrete node x is as follows: $Dom(an)$ represents the set of $\mathcal{A}$-states q associated with the copies of $\mathcal{A}$ in the run r which read the input node ν_x of $\mathcal{T}$, while for each $q \in Dom(an)$, the set of atoms $atom$ such that $(q, atom) \in an$ represents the model of $\delta(q, Lab_\mathcal{T}(\nu_x))$ selected by $\mathcal{A}$ in r on reading node ν_x in state q. Additionally, the second annotation an' in the labeling of node x keeps tracks, in case x is not the root, of the subset of the moves in the first annotation of the parent ν' of ν_x in T for which, starting from ν', a copy of $\mathcal{A}$ is sent to the current node ν_x along r. Moreover, we require that the two annotations an and an' are consistent, i.e., $an' = \emptyset$ if x is the root and $Cod(an') = Dom(an)$ otherwise. An *annotated extension* $\langle \{1, \ldots, k_\mathcal{S}\}^*, Lab_\Upsilon \rangle$ of $\langle \{1, \ldots, k_\mathcal{S}\}^*, Lab_\perp \rangle$ is *well-formed* if it satisfies the local requirements informally expressed above. We deduce the following result.

Claim 1: one can construct in singly exponential time a parity NPTA $\mathcal{P}_{wf}$ over Υ-labeled complete $k_\mathcal{S}$-trees accepting the set of *well-formed* annotated extensions of the $\perp$-completion encodings of the strategy trees of $\mathcal{G}(\mathcal{S})$. Moreover, $\mathcal{P}_{wf}$ has number of states $O(|Q| \cdot 2^{O(|Q_\mathcal{A}| \cdot |Atoms(\mathcal{A})|)})$, index 1, and transition function ρ such that $||\rho|| = O(|\Delta|)$.

In order to check that the memoryless run r of the ACG $\mathcal{A}$ over the input $\mathcal{T}$ encoded by a well-formed annotated extension $\langle \{1, \ldots, k_\mathcal{S}\}^*, Lab_\Upsilon \rangle$ of $\langle \{1, \ldots, k_\mathcal{S}\}^*, Lab_\perp \rangle$ is accepting, we proceed as follows. Let π be an infinite path of $\langle \{1, \ldots, k_\mathcal{S}\}^*, Lab_\Upsilon \rangle$ from the root which does not visit a $\perp$-labeled node. Then, $Lab_\Upsilon(\pi)$ keeps tracks of all infinite sequences of states in $Q_\mathcal{A}$ (we call $Q_\mathcal{A}$*-paths*) along r associated with the input path of the strategy tree $\mathcal{T}$ encoded by π. In particular, if $Lab_\Upsilon(\pi(i)) = (\sigma_i, an_i, an'_i)$ for all $i \geq 0$, these $Q_\mathcal{A}$-paths correspond to the sequences $q_0 q_1 \ldots$ of $Q_\mathcal{A}$-states such that for all $i \geq 0$, $q_i \in Dom(an_i)$ and $(q_i, (q_{i+1}, m, A)) \in an_i \cap an'_{i+1}$ for some $m \in \{\Box, \Diamond\}$ and set A of agents. We need to check that all these $Q_\mathcal{A}$-paths satisfy the acceptance condition of $\mathcal{A}$. Then, we first easily construct a co-parity nondeterministic word automaton $\mathcal{B}$ over Υ with $O(|Q_\mathcal{A}| \cdot |Atoms(\mathcal{A})|)$ states and index h (the index of $\mathcal{A}$) which accepts an infinite word over Υ iff it contains a $Q_\mathcal{A}$-path that does not satisfy the parity acceptance condition of $\mathcal{A}$. We now co-determinize $\mathcal{B}$, i.e., determinize it and complement it in a singly-exponential construction (Safra 1988) to obtain a deterministic parity word automaton $\mathcal{B}'$ that rejects violating $Q_\mathcal{A}$-paths. By (Safra 1988), $\mathcal{B}'$ has $(nh)^{O(nh)}$ states and index $O(nh)$, where $n = |Q_\mathcal{A}| \cdot |Atoms(\mathcal{A})|$. From $\mathcal{B}'$, we construct a standard parity nondeterministic tree automaton (parity NTA) $\mathcal{A}_{acc}$ over Υ-labeled complete $k_\mathcal{S}$-trees having $(nh)^{O(nh)}$ states and index $O(nh)$ obtained by simply running $\mathcal{B}'$ in parallel over all the branches of the input which do not visit a $\perp$-labeled node. Then, the parity NPTA $\mathcal{P}$ satisfying Theorem 3.5 is obtained by projecting out the annotation components of the input trees accepted by the intersection of the NPTA $\mathcal{P}_{wf}$ in Claim 1 with the parity NTA $\mathcal{A}_{acc}$ (recall that parity NPTA are effectively and polynomial-time closed under projection and intersection with nondeterministic tree automata (Kupferman, Piterman, and Vardi 2002)). This concludes the proof of Theorem 3.5. ☐

4 4EXPTIME–hardness of ATL* Pushdown Module-checking

In this section, we establish the following result.

Theorem 4.1. *Pushdown module-checking against* ATL* *is* 4EXPTIME*–hard even for two-player turn-based* PMS *of fixed size.*

Theorem 4.1 is proved by a polynomial-time reduction from the word problem for 3EXPSPACE–bounded Alternating Turing Machines (ATM, for short) with a binary branching degree. Formally, such a machine 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 $\delta : Q \times \Sigma \to (Q \times \Sigma \times \{\leftarrow, \rightarrow\})^2$. Configurations of $\mathcal{M}$ are words in $\Sigma^* \cdot (Q \times \Sigma) \cdot \Sigma^*$. A configuration $C = \eta\cdot(q,\sigma)\cdot\eta'$ denotes that the tape content is $\eta\cdot\sigma\cdot\eta'$, the current state (resp., input symbol) is q (resp., σ), and the reading head is at position $|\eta|+1$. From configuration C, the machine $\mathcal{M}$ nondeterministically chooses a triple (q',σ',d) in $\delta(q,\sigma) = \langle(q_l,\sigma_l,d_l),(q_r,\sigma_r,d_r)\rangle$, 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 d. We denote by $succ_l(C)$ and $succ_r(C)$ the successors of C obtained by choosing respectively the left and the right triple in $\langle(q_l,\sigma_l,d_l),(q_r,\sigma_r,d_r)\rangle$. The configuration C is accepting (resp., universal, resp., existential) if the associated state q is in F (resp., in $Q_\forall$, resp., in $Q_\exists$). Given an input $\alpha \in \Sigma^+$, a (finite) computation tree of $\mathcal{M}$ over α is a finite tree in which each node is labeled by a configuration. The root of the tree is labeled by the initial configuration associated with α. An *internal* node that is labeled by a universal configuration C has two children, corresponding to $succ_l(C)$ and $succ_r(C)$, while an internal node labeled by an existential configuration C has a single child, corresponding to either $succ_l(C)$ or $succ_r(C)$. The tree is accepting if each its leaf is labeled by an accepting configuration. An input $\alpha \in \Sigma^+$ is *accepted* by $\mathcal{M}$ if there is an accepting computation tree of $\mathcal{M}$ over α.

If the ATM $\mathcal{M}$ is 3EXPSPACE–bounded, then there is a constant $c \geq 1$ such that for each $\alpha \in \Sigma^+$, the space needed by $\mathcal{M}$ on input α is bounded by $Tower(|\alpha|^c, 3)$, where for all $n,h \in \mathbb{N}$, $Tower(n,h)$ denotes a tower of exponentials of height h and argument n (i.e, $Tower(n,0) = n$ and $Tower(n,h+1) = 2^{Tower(n,h)}$). It is well-known (Chandra, Kozen, and Stockmeyer 1981) that the acceptance problem for 3EXPSPACE–bounded ATM is 4EXPTIME-complete even if the ATM is assumed to be of fixed size.

Fix a 3EXPSPACE–bounded ATM $\mathcal{M}$ and an input $\alpha \in \Sigma^+$. Let $n = |\alpha|$. W.l.o.g. we assume that the constant c is 1 and $n > 1$. Hence, any reachable configuration of $\mathcal{M}$ over α can be seen as a word in $\Sigma^*\cdot(Q\times\Sigma)\cdot\Sigma^*$ of length $Tower(n,3)$, and the initial configuration is $(q_0,\alpha(0))\alpha(1)\ldots\alpha(n-1)\cdot(\#)^t$ where $t = Tower(n,3) - n$. Note that for an ATM configuration $C = u_1u_2\ldots u_{Tower(n,3)}$ and for all $i \in [1, Tower(n,3)]$ and $dir \in \{l,r\}$, the value u'_i of the i-th cell of $succ_{dir}(C)$ is completely determined by the values u_{i-1}, u_i and u_{i+1} (taking u_{i+1} for $i = Tower(n,3)$ and u_{i-1} for $i = 1$ to be some special symbol, say $\vdash$). We denote by $next_{dir}(u_{i-1},u_i,u_{i+1})$ our expectation for u'_i (this function can be trivially obtained from the transition function of $\mathcal{M}$). According to the previous observation, we use the set Λ of triples of the form (u_p,u,u_s) where $u \in \Sigma\cup(Q\times\Sigma)$, and $u_p,u_s \in \Sigma\cup(Q\times\Sigma)\cup\{\vdash\}$. We prove the following result from which Theorem 4.1 directly follows.

Theorem 4.2. *One can construct, in time polynomial in the size of $\mathcal{M}$ and the length n of the $\mathcal{M}$-input α, a turn-based PMS $\mathcal{S}$ and an ATL* state formula φ over the set of agents $Ag = \{sys, env\}$ such that $\mathcal{M}$ accepts α* iff *there is a strategy tree in $exec(\mathcal{G}(\mathcal{S}))$ that satisfies φ* iff *$\mathcal{G}(\mathcal{S}) \not\models_r \neg\varphi$. Moreover, the size of $\mathcal{G}(\mathcal{S})$ depends only on the size of $\mathcal{M}$.*

The rest of this section is devoted to the proof of Theorem 4.2. We first define an encoding of the ATM configurations by using the following set *Main* of atomic propositions.

$$Main := \Lambda \cup \{0,1,\forall,\exists,l,r,f\} \cup \{\mathsf{s}_1,\mathsf{s}_2,\mathsf{s}_3,\mathsf{e}_1,\mathsf{e}_2,\mathsf{e}_3\}$$

In the encoding of an ATM configuration, for each ATM cell, we record the content of the cell, the location (cell number) of the cell on the ATM tape, and the contents of the previous and next cell (if any). In order to encode the cell number, which is a natural number in $[0, Tower(n,3)-1]$, for all $1 \leq h \leq 3$, we define the notions of *h-block* and *well-formed h-block*. For $h = 1,2$, well-formed h-blocks encode integers in $[0, Tower(n,h)-1]$, while well-formed 3-blocks encode the cells of ATM configurations. In particular, for $h = 2,3$, a well-formed h-block encoding a natural number $m \in [0, Tower(n,h)-1]$ is a sequence of $Tower(n,h-1)$ well-formed $(h-1)$-blocks, where the i^{th} $(h-1)$-block encodes both the value and the position of the i^{th}-bit in the binary representation of m. Formally, a 0-block is a word of the form $\{b\}$ where $b \in \{0,1\}$ (b is the *content* of $\{b\}$). For $1 \leq h \leq 3$, an *h-block bl* is a word of the form $\{\mathsf{s}_h\}\cdot bl_0\ldots bl_t\cdot\{\tau\}\cdot\{\mathsf{e}_h\}$, where (i) $t \geq 1$, $\tau \in \{0,1\}$ if $h \neq 3$, and $\tau \in \Lambda$ otherwise (τ is the *content* of *bl*), and (ii) for all $0 \leq i \leq t$, bl_i is an $(h-1)$-block. The h-block *bl* is *well-formed* if $t = Tower(n,h-1)-1$ and whenever $h > 1$, then the $(h-1)$-block bl_i is well-formed and has *number* i for each $0 \leq i \leq t$. In this case, the *number* of *bl* is the natural number in $[0, Tower(n,h)-1]$ whose binary code is given by $b_0\ldots b_t$ where b_i is the content of the sub-block bl_i for all $0 \leq i \leq t$.

ATM configurations $C = u_1u_2\ldots u_k$ (note that here we do not require that $k = Tower(n,3)$) are then encoded by words w_C of the form $w_C = tag_1\cdot bl_1\cdot\ldots\cdot bl_k\cdot tag_2$, where $tag_1 \in \{\{l\},\{r\}\}$, for each $i \in [1,k]$, bl_i is a 3-block whose content is (u_{i-1},u_i,u_{i+1}) (where $u_0 = \vdash$ and $u_{k+1} = \vdash$), $tag_2 = \{f\}$ if C is accepting, $tag_2 = \{\exists\}$ if C is non-accepting and existential, and $tag_2 = \forall$ otherwise. The symbols l and r are used to mark a left and a right ATM successor, respectively. We also use the symbol l to mark the initial configuration. If $k = Tower(n,3)$ and for each $i \in [1,k]$, bl_i is a well-formed 3-block having number $i-1$, then we say that w_C is a *well-formed code* of C. A sequence $w_{C_1}\cdot\ldots\cdot w_{C_p}$ of well-formed ATM configuration codes is *faithful to the evolution* of $\mathcal{M}$ if for each $1 \leq i < p$, either $w_{C_{i+1}}$ is marked by symbol l and $C_{i+1} = succ_l(C_i)$, or $w_{C_{i+1}}$ is marked by symbol r and $C_{i+1} = succ_r(C_i)$.

Behaviour of the PMS $\mathcal{S}$ and encoding of accepting computation trees on α. The PMS $\mathcal{S}$ in Theorem 4.2 generates, for different environment behaviors, all the possible computation trees of $\mathcal{M}$. External nondeterminism is used in order to produce the actual symbols of each ATM configuration code. Whenever the PMS $\mathcal{S}$ reaches the end of an existential (resp., universal) guessed ATM configuration code w_C, it simulates the existential (resp., universal) choice of $\mathcal{M}$ from C by external (resp., internal) nondeterminism, and, in particular, $\mathcal{S}$ chooses a symbol in $\{l,r\}$ and marks the next guessed ATM configuration with this symbol. This ensures that, once we fix the environment behavior, we really get a tree T where each existential ATM configuration code is followed by (at least) one ATM configuration code marked by a symbol in $\{l,r\}$, and every universal configuration is followed (in different branches) by two ATM configurations codes, one marked by the symbol l and the other one marked by the symbol r. We

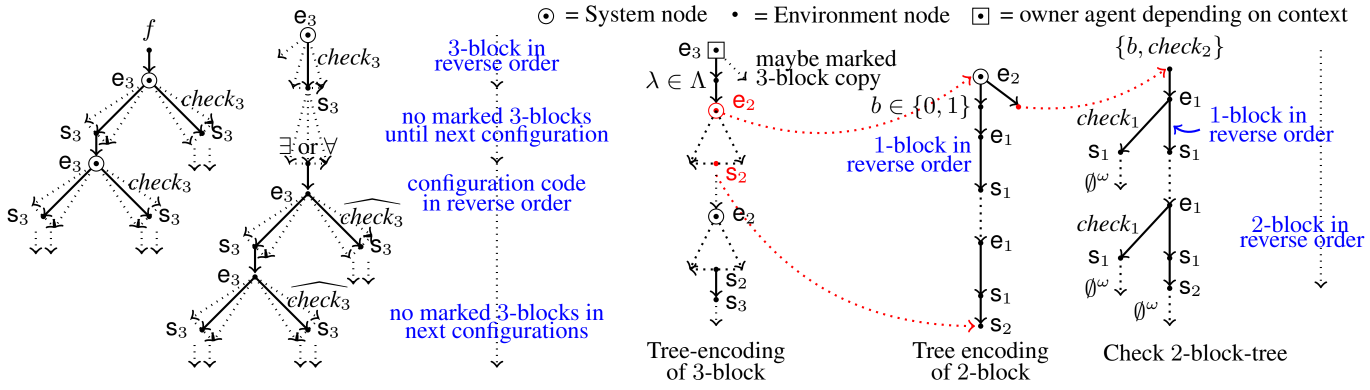

Figure 2: Subtree of the computation tree of the open PMS $\mathcal{S}$ rooted at an f-node (pop-phase)

have to check that the guessed computation tree T (corresponding to environment choices) corresponds to a legal computation tree of $\mathcal{M}$ over α. To that purpose, we have to check several properties about each computation path π of T, in particular: (i) the ATM configurations codes are well-formed (i.e., the $Tower(n,3)$-bit counter is properly updated), and (ii) π is faithful to the evolution of $\mathcal{M}$. The PMS $\mathcal{S}$ cannot guarantee by itself these requirements. Thus, these checks are performed by a suitable ATL* formula φ. However, in order to construct an ATL* formula of polynomial size, we need to 'isolate' the (arbitrary) selected path π from the remaining part of the tree. This is the point where we use the stack of the PMS $\mathcal{S}$. As the ATM configurations codes are guessed symbol by symbol, they are pushed onto the stack of the PMS $\mathcal{S}$. Whenever the end of an *accepting* computation path π (i.e., a sequence of ATM configuration codes where the last ATM configuration is accepting) is reached, the PMS by using both internal and external nondeterminism pop the entire computation path π from the stack. In this way, the PMS $\mathcal{S}$ partitions the sanity checks for π into separate branches (corresponding to the reverse of π and augmented with additional information). In particular, $\mathcal{S}$ marks by *internal* nondeterminism the content of exactly one 3-block bl_3 with the special symbol $check_3$ and, successively, (in case bl does not belong to the first configuration code of π) marks by *external* nondeterminism the content of exactly one 3-block bl'_3 with the special symbol $\widehat{check_3}$ (see Figure 2 (at left)) by ensuring that bl_3 and bl'_3 belong to two consecutive configurations codes along π. Moreover, for each 2-block bl_2 of π, $\mathcal{S}$ generates by *internal* nondeterminism a tree copy of bl_2 (*check* 2*-block-tree*). This tree (see Figure 2 (at right)) consists of a marked copy (of the reverse) of bl_2 (the content of bl_2 is marked by the special symbol $check_2$) extended with additional branches (chosen by *external* nondeterminism) which represent marked copies of the (reverse of) 1-sub-blocks bl_1 of bl_2 (the content of bl_1 is marked by the special symbol $check_1$).

Let $AP = Main \cup \{check_1, check_2, check_3, \widehat{check_3}\}$. We now formally define the AP-labeled trees associated with the *accepting* strategy trees of $\mathcal{G}(\mathcal{S})$, i.e. the strategy trees where each play from the root visits a $\{f\}$-labeled node. In the following, a 2^{AP}-labeled tree is *minimal* if the children of each node have distinct labels. A *branching-node* of a tree is a node having at least two distinct children. A *tree-code* is a *finite minimal* 2^{AP}-labeled tree $\langle T, Lab\rangle$ such that (i) for each path π from the root, $Lab(\pi)$ is a sequence of ATM configuration codes, (ii) a node x is labeled by $\{f\}$ iff x is a leaf, and (iii) each node labeled by $\{\forall\}$ has two children, one labeled by $\{l\}$ and one labeled by $\{r\}$.

Intuitively, tree-codes correspond to the maximal portions of the *accepting* strategy trees of $\mathcal{G}(\mathcal{S})$ where $\mathcal{S}$ performs push operations (push-phase). We now extend a tree-code $\langle T, Lab\rangle$ with extra nodes in such a way that each leaf x of $\langle T, Lab\rangle$ is expanded in a tree, called *check-tree* (pop-phase).

Check-trees: the definition of check-trees is based on the notion of *check* 2*-block-tree* and *simple check-tree*. The structure of a check 2-block-tree for a 2-block bl_2 is depicted in Figure 2 (at right): the branching nodes are labeled by $\{e_1\}$ and are controlled by the environment. A *partial check* 2*-block-tree* for bl_2 is obtained from the check 2-block-tree for bl_2 by pruning some choices from the branching nodes. For a sequence ν of ATM configuration codes, a *simple check-tree* for ν is a *minimal* 2^{AP}-labeled tree $\langle T, Lab\rangle$ such that

- for each path π from the root, $Lab(\pi)$ corresponds to the *reverse* of ν followed by $\emptyset^\omega$ but there is exactly one 3-block bl_3 of ν whose content is additionally marked by proposition $check_3$, and in case bl_3 does not belong to the first configuration code of ν, there is exactly one 3-block bl'_3 whose content is marked by proposition $\widehat{check_3}$; moreover, bl'_3 and bl_3 belong to two consecutive configuration codes, and bl'_3 precedes bl_3 along ν;
- for each 3-block bl_3 of ν, there is a path π from the root such that the sequence of nodes associated with bl_3 is marked by $check_3$ (i.e., all the 3-blocks of ν are checked);
- each branching-node x has label $\{e_3\}$ and two children: one labeled by $\{\lambda\}$ and the other one labeled by $\{\lambda, tag\}$ for some $\lambda \in \Lambda$ and $tag \in \{check_3, \widehat{check_3}\}$. If $tag = check_3$ (resp., $tag = \widehat{check_3}$), we say that x is a $check_3$-branching (resp., $\widehat{check_3}$-branching) node.

Finally, a *check-tree* for ν is a *minimal* 2^{AP}-labeled tree $\langle T, Lab\rangle$ which is obtained from some simple check-tree $\langle T', Lab'\rangle$ for ν by adding for each node x of T' with label $\{e_2\}$ an additional child y and a subtree rooted at y so that the subtree rooted at x obtained by removing all the descendants of x in T' is a partial check 2-block-tree for the 2-block associated with node x in T'. Thus, in a check-tree, we have four types of branching nodes: $check_3$-branching nodes and $\{\mathsf{e}_2\}$-branching nodes which are controlled by the system, and $\widehat{check_3}$-branching nodes and $\{\mathsf{e}_1\}$-branching nodes which are controlled by the environment.

Extended tree-codes: An *extended tree-code* is a minimal 2^{AP}-labeled tree $\langle T_e, Lab_e\rangle$ such that there is a tree-code $\langle T, Lab\rangle$ so that $\langle T_e, Lab_e\rangle$ is obtained from $\langle T, Lab\rangle$ by replacing each leaf x with a check-tree for the sequence of labels associated with the path of $\langle T, Lab\rangle$ leading to x. By construction and the intuitions given about the PMS $\mathcal{S}$, we easily obtain the following result.

Lemma 4.3. *One can build, in time polynomial in the size of the ATM $\mathcal{M}$, a PMS $\mathcal{S}$ over AP and $Ag = \{env, sys\}$ s.t.:*

- *the set of 2^{AP}-labeled trees $\langle T, Lab\rangle$ associated with the* accepting *strategy trees $\langle T, Lab, \tau\rangle$ in $exec(\mathcal{G}(\mathcal{S}))$ coincides with the set of extended tree-codes;*
- *for each* accepting *strategy tree $\langle T, Lab, \tau\rangle$ in $exec(\mathcal{G}(\mathcal{S}))$, the unique nodes controlled by the system in a check-subtree of $\langle T, Lab, \tau\rangle$ are the $check_3$-branching nodes and the $\{\mathsf{e}_2\}$-branching nodes.*

Construction of the ATL* formula φ in Theorem 4.2. A check-tree $\langle T, Lab\rangle$ for a sequence ν of ATM configuration codes is *well-formed* if

- *goodness:* there are no $\widehat{check_3}$-branching nodes (this means that the subtree rooted at the $\{\mathsf{s}_3\}$-node of a $check_3$-marked 3-block contains at most one $\widehat{check_3}$-marked 3-block), and each $\{\mathsf{e}_1\}$-node in a partial 2-block check-subtree has two children (i.e., all the environment choices in the $\{\mathsf{e}_1\}$-branching nodes are enabled);
- the ATM configuration codes in ν are well-formed;
- ν starts with the code of the initial configuration for α;
- *fairness:* ν is faithful to the evolution of $\mathcal{M}$ and for each path visiting a (well-formed) $check_3$-marked 3-block bl_3 and a (well-formed) $\widehat{check_3}$-marked 3-block bl'_3, bl_3 and bl'_3 have the same number.

An extended tree-code $\langle T_e, Lab_e\rangle$ is *well-formed* if each check-tree in $\langle T_e, Lab_e\rangle$ is well-formed. Evidently, there is a well-formed extended tree-code iff there is an accepting computation tree of $\mathcal{M}$ over α. We show the following result that together with Lemma 4.3 provides a proof of Theorem 4.2.

Lemma 4.4. *One can construct in time polynomial in $|AP|$ and the length n of the $\mathcal{M}$-input α, an ATL* state formula φ over AP and $Ag = \{env, sys\}$ such that for each strategy tree $\mathcal{T} = \langle T, Lab, \tau\rangle$ in $exec(\mathcal{G}(\mathcal{S}))$, $\mathcal{T}$ is a model of φ iff $\langle T, Lab\rangle$ is a well-formed extended tree-code.*

Sketched Proof. The crucial part of the proof concerns the definition of two ATL* formulas φ_{conf} and φ_{fair} satisfying the following for each *good* check-tree $\langle T, Lab\rangle$ of an accepting strategy tree of the PMS $\mathcal{S}$ of Lemma 4.3: $\langle T, Lab\rangle$ is a model of φ_{conf} (resp., φ_{fair}) iff the ATM configuration codes in $\langle T, Lab\rangle$ are well-formed (resp., $\langle T, Lab\rangle$ satisfies the *fairness* requirement). We focus on the definition of the formula φ_{fair}. By construction and the goodness requirement, for ensuring the fairness requirement, it suffices to require that for each (well-formed) $check_3$-marked 3-block bl_3 in $\langle T, Lab\rangle$ which does not belong to the first configuration code, denoted by bl'_3 the unique (well-formed) $\widehat{check_3}$-marked 3-block in the subtree rooted at the s_3-node of bl_3 and by (u_p, u, u_s) (resp., (u'_p, u', u'_s)) the content of bl_3 (resp., bl'_3), the following holds: (i) bl_3 and bl'_3 have the same number, and (ii) $u = next_l(u'_p, u', u'_s)$ if l marks the ATM configuration code of bl_3, and $u = next_r(u'_p, u', u'_s)$ otherwise.

Here, we define the ATL* formula $\varphi_=$ ensuring that bl_3 and bl'_3 have the same number. For this, we exploit the auxiliary formula $\psi_=$ in the definition of $\varphi_=$ for requiring from the current e_2-node x of the current 2-sub-block bl_2 of bl_3 that the 2-sub-block bl'_2 of bl'_3 having the same number as bl_2 has the same content as bl_2 too. Recall that in a good check-tree, the unique nodes controlled by the system are the $check_3$-branching nodes and the $\{\mathsf{e}_2\}$-nodes, and each strategy of the system selects exactly one child for each node controlled by the system. Thus, the formula $\psi_=$ asserts the existence of a strategy f_x of the player system s.t. the following holds:

1. each outcome of f_x from node x visits a node marked by $check_2$ whose parent (e_2-node) belongs to a $\widehat{check_3}$-marked 3-block. This ensures that all the outcomes get trapped in the *same* check 2-block-tree associated with some 2-block bl'_2 of bl'_3. Moreover, bl_2 and bl'_2 have the same content.
2. For each outcome π' of f_x from x which leads to a marked 1-sub-block bl'_1 (hence, a marked copy of a 1-sub-block of bl'_2), the 1-sub-block of bl_2 having the same number as bl'_1 has the same content as bl'_1 too. This ensures that bl_2 and bl'_2 have the same number.

The first (resp., second) condition is implemented by the first (resp., second) conjunct in the argument of the strategic quantifier $\langle\langle sys\rangle\rangle$ in the auxiliary subformula $\psi_=$ of $\varphi_=$.

$$\varphi_= := \bigwedge_{dir\in\{l,r\}} \mathsf{AG}\Big(\big[check_3 \wedge (\neg l \wedge \neg r)\,\mathsf{U}\,(dir \wedge \mathsf{X}(\exists \vee \forall))\big] \longrightarrow \big[(\neg \mathsf{e}_3 \wedge (\mathsf{e}_2 \to \psi_=))\,\mathsf{U}\,\mathsf{s}_3\big]\Big)$$

$$\psi_= := \langle\langle sys\rangle\rangle\Big(\mathsf{F}\big[\widehat{check_3} \wedge (\neg \mathsf{e}_3\,\mathsf{U}\,check_2)\big] \wedge \mathsf{F}check_1 \to \mathsf{X}((\neg \mathsf{e}_2 \wedge (\mathsf{e}_1 \to \mathsf{X}\eta_1))\,\mathsf{U}\,\mathsf{s}_2)\Big)$$

$$\eta_1 := \Big(\bigwedge_{i=1}^{i=n} \bigvee_{b\in\{0,1\}} ((\mathsf{X}^i\, b) \wedge \mathsf{F}(check_1 \wedge \mathsf{X}^i b))\Big) \longrightarrow \bigvee_{b\in\{0,1\}} (b \wedge \mathsf{F}(check_1 \wedge b)) \qquad \square$$

5 Conclusion

In this paper, we have addressed and carefully investigated the computational complexity of the module-checking problem of multi-agent pushdown systems (PMS) against ATL and ATL* specifications. As future work, we aim to investigate the considered problems in the setting of *imperfect information under memoryless strategies*. We recall that this setting is decidable in the finite-state case (Alur, Henzinger, and Kupferman 2002). However, moving to pushdown systems one has to distinguish whether the missing information relies in the control states, in the pushdown store, or both. We recall that in pushdown module checking only the former case is decidable for specifications given in CTL and CTL* (Aminof et al. 2013).

References

Alur, R.; Henzinger, T. A.; and Kupferman, O. 2002. Alternating-time temporal logic. *Journal of the ACM* 49(5):672–713.

Aminof, B.; Legay, A.; Murano, A.; Serre, O.; and Vardi, M. Y. 2013. Pushdown module checking with imperfect information. *Inf. Comput.* 223(1):1–17.

Aminof, B.; Kupferman, O.; and Murano, A. 2012. Improved model checking of hierarchical systems. *Inf. Comput.* 210:68–86.

Aminof, B.; Mogavero, F.; and Murano, A. 2014. Synthesis of hierarchical systems. *Sci. Comput. Program.* 83:56–79.

Ball, T., and Rajamani, S. 2000. Bebop: a symbolic model checker for boolean programs. In *7th SPIN Workshop*, LNCS 1885, 113–130. Springer.

Bouajjani, A.; Esparza, J.; and Maler, O. 1997. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *CONCUR'97*, LNCS 1243, 135–150. Springer.

Bozzelli, L., and Murano, A. 2017. On the complexity of ATL and ATL* module checking. In *GandALF'17*, EPTCS 256, 268–282.

Bozzelli, L.; Murano, A.; and Peron, A. 2010. Pushdown module checking. *Formal Methods in System Design* 36(1):65–95.

Bozzelli, L.; Murano, A.; and Peron, A. 2020. Module checking of pushdown multi-agent systems. *CoRR* abs/2003.04728.

Bozzelli, L. 2011. New results on pushdown module checking with imperfect information. In *GandALF'11*, EPTCS 54, 162–177.

Chandra, A.; Kozen, D.; and Stockmeyer, L. 1981. Alternation. *Journal of the ACM* 28(1):114–133.

Chen, T.; Song, F.; and Wu, Z. 2016. Global Model Checking on Pushdown Multi-Agent Systems. In *AAAI'16*, 2459–2465. AAAI Press.

Clarke, E., and Emerson, E. 1981. Design and synthesis of synchronization skeletons using branching time temporal logic. In *LP'81*, LNCS 131, 52–71.

Emerson, E., and Halpern, J. 1986. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *Journal of the ACM* 33(1):151–178.

Hague, M., and Ong, C. L. 2009. Winning regions of pushdown parity games: A saturation method. In *CONCUR'09*, LNCS 5710, 384–398. Springer.

Harel, D.; Rosner, R.; and Vardi, M. Y. 1990. On the power of bounded concurrency III: Reasoning about programs (preliminary report). In *LICS'90*, 478–488. IEEE Computer Society.

Jamroga, W., and Murano, A. 2014. On module checking and strategies. In *AAMAS'14*, 701–708. IFAAMAS/ACM.

Jamroga, W., and Murano, A. 2015. Module checking of strategic ability. In *AAMAS'15*, 227–235. ACM.

Kupferman, O., and Vardi, M. 1996. Module checking. In *CAV'96*, LNCS 1102, 75–86. Springer.

Kupferman, O.; Piterman, N.; and Vardi, M. 2002. Pushdown specifications. In *LPAR'02*, LNAI 2514, 262–277. Springer-Verlag.

Löding, C.; Madhusudan, P.; and Serre, O. 2004. Visibly pushdown games. In *FSTTCS'04*, LNCS 3328, 408–420. Springer.

Murano, A., and Perelli, G. 2015. Pushdown Multi-Agent System Verification. In *IJCAI'15*, 1090–1097. AAAI Press.

Murano, A.; Napoli, M.; and Parente, M. 2008. Program complexity in hierarchical module checking. In *LPAR'08*, LNCS 5330, 318–332. Springer.

Pnueli, A. 1977. The temporal logic of programs. In *FOCS'77*, 46–57. IEEE.

Queille, J., and Sifakis, J. 1981. Specification and verification of concurrent programs in Cesar. In *SP'81*, LNCS 137, 337–351. Springer.

Safra, S. 1988. On the complexity of ω-automata. In *FOCS'88*, 319–327. IEEE.

Schewe, S., and Finkbeiner, B. 2006. Satisfiability and finite model property for the alternating-time *mu*-calculus. In *CSL'06*, LNCS 4207, 591–605. Springer.

Walukiewicz, I. 1996. Pushdown processes: Games and Model Checking. In *CAV'96*, LNCS 1102, 62–74. Springer.