Pushdown Multi-Agent System Verification

Aniello Murano[1*] **and Giuseppe Perelli**[1,2 *]
[1]Università degli studi di Napoli "Federico II", [2]University of Oxford

Abstract

In this paper we investigate the model-checking problem of pushdown multi-agent systems for ATL* specifications. To this aim, we introduce *pushdown game structures* over which ATL* formulas are interpreted. We show an algorithm that solves the addressed model-checking problem in 3ExpTime. We also provide a 2ExpSpace lower bound by showing a reduction from the word acceptance problem for deterministic Turing machines with doubly exponential space.

1 Introduction

Model Checking is a well-established method widely used to verify hardware and software systems [Clarke et al., 2002]. The idea is simple and appealing: we use a mathematical model of the system we want to validate and check it over a formal specification of its desired behavior [Clarke and Emerson, 1981, Queille and Sifakis, 1981].

In the eighties, early use of model checking mainly considered *finite-state closed systems*, modeled as *Kripke structures*, and specifications given in terms of temporal-logic formulas [Pnueli, 1977]. The conceived algorithms, however, turn less appropriate in *open-system* verification as one has to take into account also the uncertainty about the agents' behavior. As a first solution, *module checking* [Kupferman et al., 2001] came out with its ability of handling the interaction between the system and an external unpredicted environment. Precisely it takes as inputs a graph partitioned in two sets (called a *module*) $\mathcal{M}$ and a formula φ, and checks whether $\mathcal{M}$ *reactively* satisfies φ, *i.e.*, no matter how the environment behaves.

Starting from the works on module checking, two significant directions have been taken in open-system verification. One concerns extending the framework to more sophisticated systems while maintaining the dichotomy system-environment states in modeling. In this context, worthy of mention is the work on *pushdown module checking* [Bozzelli et al., 2010]. This has the merit of having handled the verification of infinite-state open systems and, thanks to the fact that the infinite number of states is induced by a recursive structure of finite size, the problem turns out to be decidable and precisely 3ExpTime-complete for specifications in CTL*. Another direction has instead completely redesigned the module checking approach in order to handle the more involved scenario of multi-agent (concurrent) systems. To let the temporal-logic framework working within this setting, *Alternating-Time Temporal Logic* (ATL*, for short) [Alur et al., 2002] has been introduced. This logic generalizes CTL* by means of strategic quantifiers. ATL* formulas are interpreted over *concurrent game structures* (CGS, for short). Given an ATL* formula $\langle\langle \mathrm{A} \rangle\rangle \psi$, with A set of agents, it is satisfied over a CGS $\mathcal{G}$ if there exists a *strategy* for the agents in A such that, no matter which strategy is executed by agents not in A, the resulting outcome in $\mathcal{G}$ satisfies ψ. As for finite-state CTL* module checking, the model-checking problem for specifications in ATL* turns out to be 2ExpTime-complete. However, the two approaches are incomparable as in module checking it is possible to use nondeterministic strategies.

Despite the undoubted utility of considering, from one hand, infinite-state open-system models induced by finite-size recursive structures and, from the other hand, multi-agent specifications, to the best of our knowledge no work has been devoted to the combination of the two.

In this paper, we consider multi-agent pushdown systems and address the related model checking problem for specifications expressed in ATL*. To this aim, we first introduce *pushdown game structures* to properly model the infinite-state multi-agent system and formalize the model checking question. Then, by means of an automata-theoretic approach, we provide a 3ExpTime solution to the addressed problem. Precisely, we construct a doubly-exponential size *pushdown parity tree automaton* that collects all execution trees satisfying the ATL* formula. Then by using the fact that the emptiness of this automaton can be checked in exponential time [Kupferman et al., 2002], we get the desired result. We also provide a 2ExpSpace lower bound by showing a reduction from the word acceptance problem for a deterministic Turing machine with doubly exponential space.

*This work has been partially supported by the ERC Advanced Grant "RACE" (291528) at Oxford and the FP7 EU project 600958-SHERPA.

Related works. In recent years, model checking of pushdown systems has received a lot of attention, largely due to the ability of these systems to capture the flow of procedure of calls and returns in programs [Alur et al., 2005]. The work in this area started with Muller and Schupp, who showed that the monadic second-order theory of graphs induced by pushdown systems is decidable [Muller and Schupp, 1985]. Walukiewicz in [Walukiewicz, 1996] showed that the model checking for pushdown systems with respect to *modal μ-calculus* is ExpTime-complete. The problem remains ExpTime-complete also for CTL and LTL, while it becomes 2ExpTime-complete for CTL* [Walukiewicz, 2000, Bouajjani et al., 1997]. In [Bozzelli et al., 2010], open pushdown systems along with the module checking paradigm have been considered. This setting has been investigated under several restrictions including the imperfect-information case [Aminof et al., 2013].

Literature on model checking of ATL* is also wide. This problem has been investigated under different settings and inspired powerful formalisms for the strategic reasoning (see [Bulling, 2014], for a recent survey). Model checkers for ATL and ATL* also exist, such as **MCMAS** [Lomuscio and Raimondi, 2006, Cermák et al., 2014, Cermák et al., 2015].

Outline The rest of the paper is organized as follows. In Section 2 we introduce PGSs and provide an example to help clarifying the setting. There, we also show how a PGS can be embedded into an infinite-state CGS. In Section 3 we recall the syntax and the semantics of ATL* over CGSs and define the model-checking problem of ATL* over PGSs. In Section 4 we show that the latter can be solved in 3ExpTime by means of an automata-theoretic approach. There, we also show a 2ExpSpace-hard lower bound. Finally, in Section 5 we summarize the achieved results and discuss some future work.

2 Pushdown Game Structures

Classically, ATL* formulas are interpreted over *Concurrent Game Structures* [Alur et al., 2002]. In this paper, we instead interpret ATL* formulas over a new semantic framework, which we call *Pushdown Game Structure*. Intuitively, this new formalism provides a concurrent game structure in which a stack is added and the labeling and transition functions depend on its content. In this section, we also show that every pushdown game structure can be transformed into a suitable concurrent game structure, so providing the required interpretation of ATL* formulas over the former. However, note that the latter requires a infinite number of states, used to represent all the possible configurations the pushdown system can enter. We start with the definition of pushdown game structures.

Definition 2.1 (Pushdown Game Structure) *A* Pushdown Game Structure (PGS, for short) *is a tuple $\mathcal{P} = \langle \mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{Loc}, \Gamma, \mathsf{tr}, \mathsf{ap}, l_o \rangle$, where* AP*,* Ag*,* Ac*,* Loc *and* Γ *are finite sets of atomic propositions, agents, actions, locations, and stack alphabet, respectively,* $l_o \in \mathrm{Loc}$ *is an initial location, and* $\mathsf{ap} : \mathrm{Loc} \times \Gamma_\perp \to 2^{\mathrm{AP}}$ *is a labeling function, where* $\Gamma_\perp = \Gamma \cup \{\perp\}$ *and* $\perp$ *is the special* bottom *stack symbol not contained in* Γ. *Let* $\mathrm{Dc} \triangleq \mathrm{Ac}^{\mathrm{Ag}}$ *be the set of* decisions, i.e., *functions from* Ag *to* Ac *representing the action choices for each agent. Then,* $\mathsf{tr} : \mathrm{Loc} \times \Gamma_\perp \times \mathrm{Dc} \to \mathrm{Loc} \times \Gamma_\perp^*$ *is a* transition function *mapping a location, a stack symbol, and a decision to a location and a word in the stack alphabet.*

A pair $s = (l, \alpha) \in \mathrm{St} \triangleq \mathrm{Loc} \times (\Gamma^* \cdot \{\perp\})$ is called *state* or *configuration*. We write $\mathsf{top}(\alpha)$ for the left most symbol of α and call it the *top* of the stack content α. The PGS moves according to the transition function. This means that, if it is in the location l, the top of the stack content is γ, and the agents make a decision d, then $\mathsf{tr}(l, \gamma, d) = (l', \alpha)$ means that the execution moves to the location l' and the symbol γ is replaced with α on the top of the stack content. We assume that, if $\perp$ is popped, then it is pushed right back, and that is the only case in which it is pushed. This means that $\perp$ is always on the bottom of the stack and nowhere else. The stack containing only the symbol $\perp$ is said to be *empty*.

As the stack has no a priori bound on its size, the set St is assumed to be possibly infinite. Saying this, it turns out that PGSs are *infinite-state multi-agent systems*.

The notion of labeling and transition can be lifted to states, as follows. For a state $s = (l, \alpha)$, we define $\mathsf{ap}(s) = \mathsf{ap}(l, \mathsf{top}(\alpha))$. Moreover, for a decision $d \in \mathrm{Dc}$, we define $\mathsf{tr}((l, \gamma \cdot \alpha), d) = (l', \beta \cdot \alpha)$, with $(l', \beta) = \mathsf{tr}(l, \gamma, d)$.

Note that for a classical *pop* we write the empty word ε on the stack. To make a classical *push* one has to first put back the read top symbol and then push the required word. The transition function also allows to perform in one step a *pop-push* operation that replaces the top stack symbol with the required word.

For our convenience, we consider also *two-player turn-based one-symbol stack* games of the form $\mathcal{P} = \langle \mathrm{AP}, \{\mathtt{E}, \mathtt{A}\}, \mathrm{Loc}, \mathrm{Loc}_{\mathtt{E}}, \mathrm{Loc}_{\mathtt{A}}, R, \mathsf{ap}, l_o \rangle$ where $\mathrm{Loc}_{\mathtt{E}}$ and $\mathrm{Loc}_{\mathtt{A}}$ are the sets of locations belonging to players E and A, respectively, and $R \subseteq (\mathrm{Loc} \times \{\gamma, \perp\}) \times (\mathrm{Loc} \times \{\mathtt{push}, , \mathtt{pop}\})$, where γ is the only alphabet symbol of the stack, and push, pop, and are the push, pop, and null operation on the stack. If $(l, x, l', \mathtt{op}) \in R$, then, for each configuration (l, α) with $\mathsf{top}(\alpha) = x$ we can move to the configuration (l', α') with α' being the string obtained from α by applying the stack operation op. At each configuration (l, α) of the game, the owner of the location l can pick a successor, according to the relation R. It is not hard to see that *two-player turn-based one-symbol stack* games are special cases of PGSs

To get familiar with PGSs, we give an example.

Example 2.1 (Pushdown scheduler) *Take a system consisting of two processes* a *and* b *that may access to a common resource via the respective requests* $\mathtt{r_a}$ *and* $\mathtt{r_b}$ *and a scheduler* s *that can grant in a* LIFO *order the processes requests, all memorized into a stack. As model we use a* PGS $\mathcal{P} = \langle \mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{Loc}, \Gamma, \mathsf{tr}, \mathsf{ap}, l_o \rangle$, *with* $\mathrm{AP} = \{\mathtt{r_a}, \mathtt{r_b}, \mathtt{g_a}, \mathtt{g_b}, \mathtt{a}, \mathtt{b}, \mathtt{e}\}$*,* $\mathrm{Ag} = \{\mathtt{s}, \mathtt{a}, \mathtt{b}\}$*,* $\mathrm{Ac} = \{\mathtt{a}, \mathtt{b}, \mathtt{un}, 0, 1\}$*,* $\mathrm{Loc} = \{l_o, l_a, l_b, l_{un}\}$*, and* $\Gamma = \{\mathtt{r_a}, \mathtt{r_b}\}$.

The scheduler $\mathtt{s}$ *controls the location* l_o *by means of the actions* $\mathtt{a}$, $\mathtt{b}$, *and* $\mathtt{un}$, *standing for "*$\mathtt{a}$ *can make a request", "*$\mathtt{b}$ *can make a request", and "the system can unload the stack requests", respectively. Accordingly, they lead to the locations* $l_{\mathtt{a}}$, $l_{\mathtt{b}}$, *and* $l_{\mathtt{un}}$. *On* $l_{\mathtt{a}}$ *and* $l_{\mathtt{b}}$, *the agents* $\mathtt{a}$ *and* $\mathtt{b}$, *respectively, can either make a request via action* 1 *or skip it with action* 0. *In the former case, the request is recorded into the stack by writing the symbol* $\mathtt{r_x}$, *for* $\mathtt{x} \in \{\mathtt{a}, \mathtt{b}\}$; *otherwise, in the latter case there is no operation over the stack. Finally, the location* $l_{\mathtt{un}}$ *triggers the granting phase by emptying the stack. During this phase, neither* $\mathtt{a}$ *nor* $\mathtt{b}$ *can make any further request. This can be seen as a legitimate constraint by thinking how classical synchronizing and backup systems are designed.*

The labeling function, for all $\gamma \in \Gamma_\perp$ *and* $\mathtt{x} \in \{\mathtt{a}, \mathtt{b}\}$, *is defined as follows:* $\mathsf{ap}(l_o, \perp) = \emptyset$, $\mathsf{ap}(l_o, \mathtt{r_x}) = \{\mathtt{x}\}$, $\mathsf{ap}(l_{\mathtt{x}}, \gamma) = \{\mathtt{x}\}$, $\mathsf{ap}(l_{\mathtt{un}}, \mathtt{r_x}) = \{\mathtt{g_x}\}$, *and* $\mathsf{ap}(l_{\mathtt{un}}, \perp) = \{\mathtt{e}\}$. *Intuitively, propositions* $\mathtt{a}$ *and* $\mathtt{b}$ *means that agents* $\mathtt{a}$ *and* $\mathtt{b}$ *are authorized to make a request, respectively. The proposition* $\mathtt{r_x}$, *instead, occurs when the corresponding request has been just made by agent* $\mathtt{x}$. *On the other hand, the proposition* $\mathtt{g_x}$ *occurs when the request* $\mathtt{r_x}$ *has been just granted. Finally,* $\mathtt{e}$ *indicates that the unloading phase is terminated and so the stack is empty.*

The transition function tr *is described directly in Figure 1. The labeling of the edges have the following meaning. First, note that it is composed of two parts separated by a semi-column. The left part represents the decision of the agents, given in the order* $\mathtt{s} < \mathtt{a} < \mathtt{b}$. *The right part represents the stack operation. As an example, the label* $*1*; push(\mathtt{r_a})$ *says that agent* $\mathtt{a}$ *is making a request* $\mathtt{r_a}$ *and the symbol* $\mathtt{r_a}$ *is pushed on the stack, where the symbol* $*$ *denotes any possible action for the other agents.*

The nodes represent all possible states. Note that, for the locations $l_{\mathtt{a}}$ *and* $l_{\mathtt{b}}$ *we have collapsed the two possible configurations with* $\beta = \perp$ *and* $\beta \neq \perp$ *since the transition over them does not depend on the stack content.*

Finally, observe that the stack is unbounded and so an execution might generate an infinite number of distinguished states. Also, observe that the stack is fundamental to keep track of the order in which the requests appear.

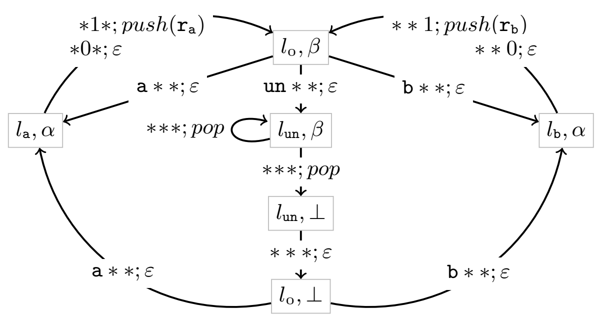

Figure 1: A Pushdown system scheduler.

To correctly interpret ATL* formulas over PGSs, we show that a PGS can be represented as an infinite-state concurrent game structure, whose definition follows. Note that we use the one reported in [Mogavero et al., 2014].

Definition 2.2 (Concurrent Game Structures) *A* concurrent game structure *(*CGS*, for short) is a tuple* $\mathcal{G} \triangleq \langle \mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, \mathsf{tr}, \mathsf{ap}, s_o \rangle$, *where* AP, Ag, *and* Ac *are as in* PGS. St *is an enumerable non-empty set of* states, $s_o \in \mathrm{St}$ *is an* initial state, *and* $\mathsf{ap} : \mathrm{St} \to 2^{\mathrm{AP}}$ *is a* labeling function *mapping each state to a set of atomic propositions true in that state. Finally,* $\mathsf{tr} : \mathrm{St} \times \mathrm{Dc} \to \mathrm{St}$ *is a* transition function *mapping pairs of states and decisions to states, where the set* Dc *is as in* PGS.

Clearly, a PGS $\mathcal{P} = \langle \mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{Loc}, \Gamma, \mathsf{tr}, \mathsf{ap}, l_o \rangle$ can be suitably turned into a CGS $\mathcal{G}_{\mathcal{P}} = \langle \mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, \mathsf{tr}, \mathsf{ap}, s_o \rangle$, where $\mathrm{St} = \mathrm{Loc} \times \Gamma_\perp^*$, $s_o = (l_o, \perp)$, and the functions ap and tr are the lifting on states of the corresponding functions in $\mathcal{P}$. Intuitively, the states of $\mathcal{G}$ are used to implicitly represent both the current location and store the stack content. Despite this, it is important to observe that, while a PGS has a finite number of control locations, the corresponding CGS necessarily has an infinite number of control states, as the number of different stack contents is unbounded.

We conclude this section by briefly recalling the classical notions of *track*, *path*, *strategies* and *assignments*, which are required for the semantics of ATL* (see [Mogavero et al., 2014], for more). Intuitively, tracks and paths are legal sequences of reachable states, respectively seen as partial and complete descriptions of possible outcomes over a CGS. Formally, a *track* (resp., *path*) in a CGS is a finite (resp., an infinite) sequence of states $\rho \in \mathrm{St}^*$ (resp., $\pi \in \mathrm{St}^\omega$) such that, for all $i \in [0, |\rho| - 1[$ (resp., $i \in \mathbb{N}$), there is a decision $d \in \mathrm{Dc}$ with $(\rho)_{i+1} = \mathsf{tr}((\rho)_i, d)$ (resp., $(\pi)_{i+1} = \mathsf{tr}((\pi)_i, d)$). The set $\mathrm{Trk} \subseteq \mathrm{St}^+$ (resp., $\mathrm{Pth} \subseteq \mathrm{St}^\omega$) contains all non-empty tracks (resp., paths). Moreover, $\mathrm{Trk}(s) \triangleq \{\rho \in \mathrm{Trk} : (\rho)_0 = s\}$ (resp., $\mathrm{Pth}(s) \triangleq \{\pi \in \mathrm{Pth} : (\pi)_0 = s\}$) denotes the subsets of tracks (resp., paths) starting at a state s.

A *strategy* for an agent is a scheme containing all choices of actions, depending on the current outcome. Formally, a strategy in a CGS is a function $\mathsf{f} : \mathrm{Trk} \to \mathrm{Ac}$ that maps each non-empty track to an action. The set Str contains all strategies. For a given subset $\mathrm{A} \subseteq \mathrm{Ag}$ of agents, an *assignment* over A is a partial function $\chi_{\mathrm{A}} : \mathrm{Ag} \rightharpoonup \mathrm{Str}$, mapping each agent in A to a strategy. By Asg we denote the set of assignments. A path is *compatible* with an assignment χ_{A} if it is obtained by agents in A using strategies in χ_{A}. More formally, for a given set $\mathrm{A} \subseteq \mathrm{Ag}$ and an assignment χ_{A} over A, we say that a path π is compatible with χ_{A} if, for all $i \in \mathbb{N}$ it holds that $(\pi)_{i+1} = \mathsf{tr}((\pi)_i, d)$, for some $d \in \mathrm{Dc}$ with $d(a) \triangleq \chi_{\mathrm{Asg}}(a)((\pi)_{\leq i})$, for each $a \in \mathrm{A}$. By $\mathsf{play}(\chi_{\mathrm{A}}, s)$ we denote the set of paths starting from s that are compatible with χ_{A}. Note that, for an assignment χ_{Ag} over the full set Ag of agents, there exists only one compatible path. In this case, by abuse of notation, we denote it with $\mathsf{play}(\chi_{\mathrm{Ag}}, s)$.

3 ATL*

In this section, we recall the syntax of ATL* and introduce its semantics over PGS via its representation in terms of CGS (with infinite states). We start with the definition of ATL* syntax.

Definition 3.1 (ATL* **Syntax**) ATL* *formulas are built inductively from the set of atomic propositions* AP *and agents* Ag*, by using the following grammar, where* $p \in \mathrm{AP}$ *and* $\mathrm{A} \subseteq \mathrm{Ag}$*:*

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{X}\varphi \mid \varphi \mathtt{U} \varphi \mid \langle\!\langle \mathrm{A} \rangle\!\rangle \varphi.$$

As syntactic sugar we also use $\varphi_1 \vee \varphi_2 \triangleq \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \triangleq \neg\varphi_1 \vee \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2 \triangleq (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$, $[\![\mathrm{A}]\!]\varphi \triangleq \neg\langle\!\langle \mathrm{A} \rangle\!\rangle\neg\varphi$, $\mathtt{F}\varphi \triangleq \mathtt{tU}\varphi$, and $\mathtt{G}\varphi \triangleq \neg\mathtt{F}\neg\varphi$.

A *sentence* is a Boolean combination of ATL* formulas of the form $\langle\!\langle \mathrm{A} \rangle\!\rangle\psi$. Intuitively, $\langle\!\langle \mathrm{A} \rangle\!\rangle\psi$ means that each agent in A has a strategy such that, whatever the other agents do, the resulting play satisfies ψ.

We now provide some examples of ATL* formulas that will be useful in the sequel. Precisely, we consider $\varphi_1 = \langle\!\langle \{\mathtt{s}\} \rangle\!\rangle(\mathtt{GFa} \wedge \mathtt{GFb} \wedge \mathtt{GFe})$, $\varphi_2 = \langle\!\langle \emptyset \rangle\!\rangle(\mathtt{GFa} \wedge \mathtt{GFb} \wedge \mathtt{GFe})$ and $\varphi_3 = \langle\!\langle \emptyset \rangle\!\rangle((\mathtt{r_a} \wedge \mathtt{X}(\neg\mathtt{eUr_b})) \rightarrow (\mathtt{Fg_b} \rightarrow \mathtt{XFg_a}))$ over the sets AP and Ag given in Example 2.1. The formula φ_1 states that agent $\mathtt{s}$ has a way to let propositions $\mathtt{a}$, $\mathtt{b}$, and $\mathtt{e}$ to occur infinitely often. The formula φ_2 states that, no matters how the agents behave, propositions $\mathtt{a}$, $\mathtt{b}$, and $\mathtt{e}$ occur infinitely often. Finally, the formula φ_3 states that, whenever a request $\mathtt{r_b}$ occurs after an $\mathtt{r_a}$ one in the same loading phase, then, if $\mathtt{r_b}$ is eventually granted, then $\mathtt{r_a}$ is granted later on as well.

We now provide the semantics of ATL*.

Definition 3.2 (ATL* **Semantics**) *For a* CGS $\mathcal{G} = \langle \mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, \mathsf{tr}, \mathsf{ap}, s_o \rangle$ *and a path* $\pi \in \mathrm{Pth}$*, the modeling relation* $\mathcal{G}, \pi \models \varphi$ *is inductively defined as follows:*

- *The atomic and boolean cases are defined as usual;*
- $\mathcal{G}, \pi \models \langle\!\langle \mathrm{A} \rangle\!\rangle\varphi$ *if there is an assignment* χ_{A} *such that* $\mathcal{G}, \pi' \models \varphi$*, for all* $\pi' \in \mathsf{play}(\chi_{\mathrm{A}}, (\pi)_0)$*;*
- $\mathcal{G}, \pi \models \mathtt{X}\varphi$ *if* $\mathcal{G}, (\pi)_{\geq 1} \models \varphi$*;*
- $\mathcal{G}, \pi \models \varphi_1 \mathtt{U} \varphi_2$ *if there exists* $j \in \mathbb{N}$ *such that* $\mathcal{G}, (\pi)_{\geq i} \models \varphi_1$*, for all* $i < j$*, and* $\mathcal{G}, (\pi)_{\geq j} \models \varphi_2$*.*

For a sentence φ and two paths π_1, π_2 with $(\pi_1)_0 = (\pi_2)_0$, it holds that $\mathcal{G}, \pi_1 \models \varphi$ iff $\mathcal{G}, \pi_2 \models \varphi$. Indeed, according to the semantics of the existential quantification $\langle\!\langle \mathrm{A} \rangle\!\rangle\psi$, the only element of the path to take into account is the first one. For this reason, for a sentence φ we write $\mathcal{G}, s \models \varphi$ if $\mathcal{G}, \pi \models \varphi$ for some $\pi \in \mathrm{Pth}(s)$. Finally, we say that $\mathcal{G}$ satisfies φ, and write $\mathcal{G} \models \varphi$, if $\mathcal{G}, s_o \models \varphi$.

To get familiar with the semantics, consider the PGS $\mathcal{P}$ given in Example 2.1 and the formulas φ_1, φ_2, and φ_3 given above. It is easy to see that $\mathcal{G}_{\mathcal{P}} \models \varphi_1$. Indeed, the strategy f_1 that allows the scheduler to pick infinitely often the actions $\mathtt{a}$, $\mathtt{b}$, and $\mathtt{un}$, makes all the generated paths to satisfy $\mathtt{GFa} \wedge \mathtt{GFb} \wedge \mathtt{GFe}$. On the other hand, it is easy to see that $\mathcal{G}_{\mathcal{P}} \not\models \varphi_2$. Indeed, the strategy f_2 for $\mathtt{s}$ such that $\mathsf{f}_2(\rho) = \mathtt{a}$, for all $\rho \in \mathrm{Trk}$, never makes $\mathtt{b}$ and $\mathtt{e}$ to occur in the generated paths. Finally, we have that $\mathcal{G}_{\mathcal{P}} \models \varphi_3$.

Definition 3.3 (Model-checking) *For a given* PGS $\mathcal{P}$ *and an* ATL* *formula* φ*, the* model-checking problem *is to decide whether* $\mathcal{G}_{\mathcal{P}} \models \varphi$*.*

4 Model Checking

In this section, we provide a 3EXPTIME upper-bound and a 2EXPSPACE lower-bound for the model-checking problem of ATL* over PGS.

4.1 Upper-bound Complexity

For the upper bound we use an automata-theoretic approach. We start with some notation and the definition of nondeterministic pushdown automata. See [Kupferman et al., 2002, Kupferman et al., 2000b] for more.

For a given set $\mathrm{D} \subseteq \mathbb{N}$, a D-tree T is a prefix closed subset of D^*, *i.e.*, a set in which, if $\mathtt{x} \cdot \mathtt{d} \in \mathrm{T}$ then $\mathtt{x} \in \mathrm{T}$. The elements of T are called *nodes* and the empty word ε is called the *root* of T. For $x \in \mathrm{T}$, the set of *children* of x is $\mathit{children}(\mathrm{T}, x) \triangleq \{x \cdot i \in \mathrm{T} : i \in \mathrm{D}\}$. For $x, y \in \mathrm{T}$, we write $x \prec y$ to mean that x is a proper prefix of y, *i.e.*, there exists $z \in \mathrm{D}^*$ such that $x \cdot z = y$. For $x \in \mathrm{T}$, a path in T from x is a minimal set $\pi \subseteq \mathrm{T}$ such that $x \in \pi$ and for each $y \in \pi$ such that $\mathit{children}(\mathrm{T}, y) \neq \emptyset$, there exist exactly one node in $\mathit{children}(\mathrm{T}, y)$ belonging to π. For an alphabet Σ, a Σ-labeled tree is a pair $\mathcal{T} = \langle \mathrm{T}, \mathsf{V} \rangle$ where T is a tree and $\mathsf{V} : \mathrm{T} \rightarrow \Sigma$ maps each node in T to an element in Σ. In the following, we mainly consider Σ to be the set power set 2^{AP} of atomic propositions AP.

A *Nondeteriministic Pushdown Tree Automata* (PD-NTA, for short), over Σ-labeled trees, is a tuple $\mathcal{A} = \langle \Sigma, \Gamma, \mathrm{Q}, q_o, \alpha_0, \delta, \mathrm{F} \rangle$, where Σ and Γ are *finite* input and stack alphabet sets, Q is a finite set of states, q_o is an initial state, $\alpha_0 \in \Gamma^* \cdot \bot$ is an initial stack content, $\delta : \mathrm{Q} \times \Sigma \times \Gamma_\bot \rightarrow 2^{2^{(\mathrm{Q} \times \Gamma^*)}}$ is a transition function such that, for all $(q, \sigma, \gamma) \in \mathrm{Q} \times \Sigma \times \Gamma_\bot$, $\delta(q, \sigma, \gamma)$ is a *finite* set, and F is an acceptance condition over Q.

When the automaton is in a state q, reading an input node x labeled with $\sigma \in \Sigma$, and the stack contains a word $\gamma \cdot \alpha$ in $\Gamma_\bot^*$, it chooses, for some $k \in \mathbb{N}$, a tuple $\langle (q_1, \beta_1), \ldots, (q_k, \beta_k) \rangle \in \delta(q, \sigma, \gamma)$ and splits in k copies such that, for each $1 \leq i \leq k$, the i-th copy is in the state q_i and the stack content is updated by removing γ and pushing β_i. Then, it reads the node $x \cdot i$ of the tree.

A run of a PD-NTA on a Σ-labeled tree $\mathcal{T} = \langle \mathrm{T}, \mathsf{V} \rangle$ is a $(\mathrm{Q} \times \Gamma_\bot^*)$-labeled tree $\langle \mathrm{T}, \mathsf{r} \rangle$ such that $\mathsf{r}(\varepsilon) = (q_o, \alpha_0)$ and for each $x \in \mathrm{T}$ with $\mathsf{r}(x) = (q, \gamma \cdot \alpha)$, there is a tuple $\langle (q_1, \beta 1), \ldots, (q_k, \beta_k) \rangle \in \delta(q, \mathsf{V}(x), \gamma)$ for some $k \in \mathbb{N}$, such that, for all $1 \leq i \leq k$, $\mathsf{r}(x \cdot i) = (q_i, \beta_i \cdot \alpha)$ if $\gamma \neq \bot$, and $\mathsf{r}(x \cdot i) = (q_i, \beta_i \cdot \bot)$, otherwise.

Given a path π starting from ε, by $\inf_{\mathsf{r}}(\pi)$ we denote the subset of states q such that there are infinitely many $x \in \pi$ such that $\mathsf{r}(x) \in \{q\} \times \Gamma_\bot^*$. A path satisfies a *parity* condition $\mathrm{F} = \{\mathrm{F}_1, \ldots, \mathrm{F}_k\}$, with $\mathrm{F}_i \subseteq \mathrm{F}_{i+1}$, for all $i < k$, and $\mathrm{F}_k = \mathrm{Q}$, if the minimum index $1 \leq i \leq k$ such that $\inf_{\mathsf{r}}(\pi) \cap \mathrm{F}_i \neq \emptyset$ is even. A run $\langle \mathrm{T}, \mathsf{r} \rangle$ is *accepting* if every path satisfies the acceptance condition. The PD-NTA $\mathcal{A}$ accepts an input tree $\langle \mathrm{T}, \mathsf{V} \rangle$ iff there is an accepting run of $\mathcal{A}$ over it. The language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, contains all the trees accepted by $\mathcal{A}$. The emptiness problem for PD-NTA is to decide, for a given $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A}) = \emptyset$. In [Kupferman et al., 2002] it is reported the following.

Theorem 4.1 *The emptiness problem for a parity* PD-NTA *is* ExpTime-complete.

In several branching-time temporal-logic verification settings, the automata-theoretic approach has been fruitfully applied. Very close to our case are the procedures deployed for model checking pushdown systems over CTL* specifications [Bouajjani et al., 1997, Bozzelli et al., 2010] and finite-state CGSs over ATL* specifications [Alur et al., 2002]. The former is a top-down procedure that first builds an automaton accepting all the trees that satisfy the formula and then checks for the membership problem of the tree unwinding of the pushdown model. Precisely, to get a tight complexity, it starts with a single-exponential alternating [1] parity tree automaton and the membership problem results in a special alternating pushdown tree automaton named *one-letter*, with no blow-up in size, whose emptiness can be checked in exponential-time[2] resulting in an overall doubly-exponential time solution. The procedure for ATL*, instead, uses a doubly-exponential bottom-up approach based on the idea of labeling each state of the structure with subformulas true in that state. In our setting we can neither proceed with the membership problem nor use a bottom-up procedure. Indeed, because of ATL*, we need to consider not just the unwinding of the model but the tree execution induced by the player existentially quantified in the formula. Moreover, because of the possible infinite number of configurations induced by the PGS, a bottom-up procedure could never terminate. For this reason, we use a top-down approach that constructs a doubly exponential PD-NTA that simultaneously checks whether a tree is an execution of the structure and a model of the formula. As far as we know, this is the first top-down automata-theoretic approach exploited for ATL*. Some details about this automata construction are reported in the following.

Theorem 4.2 *The model-checking problem for* ATL* *on* PGS *can be solved in 3*ExpTime.

Proof sketch: We give an intuition behind the automata construction by providing some details on how to extend the one introduced in [Bouajjani et al., 1997] used to solve the model-checking problem for branching-time specifications over pushdown systems. The mentioned approach starts with a tree automaton accepting all tree models of a formula φ, namely the formula automaton $\mathcal{A}_\varphi$, over which one can build a PD-NTA $\mathcal{A}_{\mathcal{P},\varphi}$ accepting the unwinding of the pushdown structure $\mathcal{P}$ iff it is contained in the language of the formula automaton $\mathcal{A}_\varphi$. To handle ATL*, one can start with a doubly-exponential parity tree automaton as an adaptation of the one provided in [Schewe, 2008][3]. Moreover, in order to correctly evaluate the formula over a PGS we need not just to consider the unwinding of the structure but rather the execution trees induced by the formula and precisely from the players existentially quantified in it. This results in selecting at each node subsets of children upon the choices of the players. As the number of these subsets is linear in the number of the decisions of the structure, the overall size of the PD-NTA we construct remains doubly-exponential. Thus, from Theorem 4.1 we derive a 3ExpTime procedure. $\square$

4.2 A Lower-bound Complexity

In this section, we show that the model-checking problem for ATL* over PGSs is 2ExpSpace-hard by means of a reduction from the word acceptance problem for a deterministic Turing machine with doubly exponential space. Such reduction is inspired by the one provided in [Vester, 2014] for one-counter games.

Let $\mathcal{T} = \langle \mathrm{Q}, q_0, \Sigma, \delta, q_F \rangle$ be a Turing machine that uses at most 2^{2^n} cells on an input w of length n where, Q is the set of control states, q_0 and q_F are the initial and final states, respectively, $\Sigma = \{0, 1, a, r, \sharp\}$ is the finite alphabet set, and $\delta : \mathrm{Q} \times \Sigma \to \mathrm{Q} \times \Sigma \times \{-1, 0, +1\}$ is the (deterministic) transition function. For our convenience, if $\delta(q, a) = (q', a', x)$ we write $\delta_1(q, a) = q'$, $\delta_2(q, a) = a'$, and $\delta_3(q, a) = x$, respectively. The input set of $\mathcal{T}$ is given by $\Sigma_\mathrm{I} = \Sigma \setminus \{\sharp\}$. From this, we can construct a PGS $\mathcal{P}_{\mathcal{T},w}$ and an ATL* formula φ such that $\mathcal{T}$ accepts w iff $\mathcal{P}_{\mathcal{T},w} \models \varphi$. To do this, we need some auxiliary notation. First, *w.l.o.g.*, we can assume that $\mathcal{T}$ always accepts when the symbol a is read, and always reject when the symbol r is read. Moreover, we can assume that $\mathcal{T}$ always halts in the position 1 of the tape and that there are two additional cells at the ends, numbered with 0 and $2^{2^n} + 1$ containing the symbol a. Let $\Delta = \Sigma \cup (\mathrm{Q} \times \Sigma)$. Then a configuration is a sequence in $\Delta^{2^{2^n}+2}$ containing exactly one element in $\mathrm{Q} \times \Sigma$. Since $\mathcal{T}$ is deterministic, then there is a unique run $\mathcal{C}_0^w \cdot \mathcal{C}_1^w \cdot \ldots$ of computations starting from $\mathcal{C}_0^w = a \cdot (q_0, w_0) \cdot \ldots w_n \cdot \sharp \cdot \ldots \cdot \sharp \cdot a$. $\mathcal{C}_i^w(j)$ denotes the j-th symbol of the i-th configuration in the computation. Observe that, given the three elements $\mathcal{C}_i^w(j-1)$, $\mathcal{C}_i^w(j)$, and $\mathcal{C}_i^w(j+1)$, then the symbol $\mathcal{C}_{i+1}^w(j)$ is uniquely determined, according to the definition of transition function. Then, for $d \in \Delta$, by $Pre(d)$ we denote the set of triples (d_1, d_2, d_3) such that $d_1 = \mathcal{C}_i^w(j-1)$, $d_2 = \mathcal{C}_i^w(j)$, $d_3 = \mathcal{C}_i^w(j+1)$, and $d = \mathcal{C}_{i+1}^w(j)$.

At this point, we consider the auxiliary *two-player turn-based one-symbol stack* game $\mathcal{R}_{\mathcal{T},w} = \langle \mathrm{AP}, \{\mathtt{E}, \mathtt{A}\}, \mathrm{Loc}, \mathrm{Loc}_\mathtt{E}, \mathrm{Loc}_\mathtt{A}, R, \mathsf{ap}, l_0 \rangle$ where:

- $\mathrm{Loc} = ([0, 2^{2^n} + 1] \times (\Delta \cup \Delta^3)) \cup \{l_0, l_z, l_r, l_F\}$;
- $\mathrm{Loc}_\mathtt{E} = ([0, 2^{2^n} + 1] \times \Delta) \cup \{l_0\}$;
- $\mathrm{Loc}_\mathtt{A} = ([0, 2^{2^n} + 1] \times \Delta^3) \cup \{l_z, l_r, l_F\}$;

[1] Automata having as transition relation a positive Boolean combination of states and directions [Kupferman et al., 2000b].

[2] Recall that in general the emptiness check for alternating pushdown automata is undecidable [Kupferman et al., 2002].

[3] In [Schewe, 2008] it is given a single-exponential alternating automaton that can be easily translated into a nondeterministic one with a single exponential-time blowup.

- R is the smallest relation such that:

– $(l_0, x, l_0, \texttt{push})$, for $x \in \{\bot, \gamma\}$;
– $(l_0, x, (1, (q_F, a)), \texttt{null})$ for $x \in \{\bot, \gamma\}$;
– $((j, d), \gamma, (j, (d_1, d_2, d_3)), \texttt{null})$ for all $(d_1, d_2, d_3) \in Pre(d)$ and $j \in [1, 2^{2^n}]$;
– $((j, a), x, l_F, \texttt{null})$ for all $j \in [1, 2^{2^n}]$;
– $((j, d), x, l_r, \texttt{null})$ if $j = 0$ or $j = 2^{2^n} + 1$;
– $((j, d), x, l_z, \texttt{null})$ if $\mathcal{C}_0^w(j) = d$;
– $(l_z, \gamma, l_F, \texttt{null})$;
– $(l_z, \gamma, l_r, \texttt{pop})$;
– $((j, (d_1, d_2, d_3)), \gamma, (j-1, d_1), \texttt{pop})$, for all $j \in [0, 2^{2^n}]$ and $d_1, d_2, d_3 \in \Delta$;
– $((j, (d_1, d_2, d_3)), \gamma, (j, d_2), \texttt{pop})$, for all $j \in [0, 2^{2^n}]$ and $d_1, d_2, d_3 \in \Delta$;
– $((j, (d_1, d_2, d_3)), \gamma, (j+1, d_3), \texttt{pop})$, for all $j \in [0, 2^{2^n}]$ and $d_1, d_2, d_3 \in \Delta$.

Intuitively, player E pushes the symbol γ into the stack a number of times that corresponds to the length of the computation accepting w. After this, the game starts from the configuration $(1, (q_F, a), \gamma)$ and proceeds backway along the computation of $\mathcal{T}$ over w. In the configurations with locations of the form (j, d), player E selects a possible predecessor triple (d_1, d_2, d_3) of d. At this point, player A selects one of the elements in (d_1, d_2, d_3), while a pop operation is performed on the stack. Finally, if the stack is empty and the location (j, d) is such that $\mathcal{C}_0^w(j) = d$, then player E can move to configuration $(l_z, \bot)$, from which player A is forced to move in location l_F, since the transition $(l_z, \gamma, l_r, \texttt{pop})$ on empty stacks is deactivated. It is not hard to see that w is accepted by $\mathcal{T}$ iff player E can force the game $\mathcal{R}_{\mathcal{T},w}$ to reach l_F.

This reasoning allows us to reduce the accepting problem to a reachability game played on $\mathcal{R}_{\mathcal{T},w}$, which is of size doubly exponential *w.r.t.* to $\mathcal{T}$, that can be specified by the ATL* formula $\langle\langle \texttt{E} \rangle\rangle \texttt{Fp}$, where p is the proposition labeling all the configurations having l_F as location. Now, having $\mathcal{R}_{\mathcal{T},w}$ in mind, we can build a PGS $\mathcal{P}_{\mathcal{T},w}$ and an ATL* formula φ such that $\mathcal{P}_{\mathcal{T},w} \models \varphi$ iff $\mathcal{T}$ accept w.

The construction of $\mathcal{P}_{\mathcal{T},w}$ is essentially a modification of $\mathcal{R}_{\mathcal{T},w}$ in which the position of the head on the tape is encoded by a suitable LTL formula ψ, rather than the set of states. This allows such a model to have polynomial size *w.r.t.* $\mathcal{T}$ and w. The way the formula ψ works is folklore and completely described in [Bozzelli et al., 2005, Kupferman et al., 2000a]. We omit it here due to the lack of space.

Finally, we can prove that $\mathcal{P}_{\mathcal{T},w} \models \langle\langle \texttt{E} \rangle\rangle (\psi \wedge \texttt{Fp})$ iff $\mathcal{T}$ accepts w, from which we derive the following theorem.

Theorem 4.3 *The model-checking problem for* ATL* *over* PGS *is in 2*EXPSPACE-HARD.

5 Conclusion

In the last years, *open* pushdown models have received a lot of attention from the formal verification community, largely due to their ability to capture the control-flow of procedure calls and returns in reactive systems [Alur et al., 2005]. In several settings, the use of pushdown models allows to verify the correctness of infinite-state systems with a decidable complexity [Piterman and Vardi, 2004, Kupferman et al., 2002, Song and Touili, 2014]. As far as we know, all the work so far has concentrated on models with at most two-agents and with respect to specifications given in terms of classic temporal logics [Abdulla et al., , Chatterjee and Velner, 2012, Bozzelli et al., 2010].

In this paper, we have introduced *multi-agent pushdown game structures* to model more involved infinite-state scenarios (as induced by a recursive structure) in which several agents can cooperate or act in an adversarial way in order to achieve a certain goal. As main contribution related to these structures we have introduced and studied the model checking problem with respect to the logic ATL* and showed that this problem can be solved in 3EXPTIME. We recall that the same complexity holds also for pushdown module checking with respect to specifications given in CTL*. The latter is a special two-player setting, where one of the player, the environment, can also use nondeterministic strategies. We also provide a non tight 2EXPSPACE lower bound. Our conjecture is that the investigated problem is 3EXPTIME-COMPLETE. We leave this as future work.

On some extent, the high complexity of the addressed problem relies on the fact that the rich formalisms of pushdown models and ATL* specification we combine are complex by themselves. While this allowed us to provide a result for a very general framework, the overall complexity can be easily reduced by considering opportune restrictions on both sides. Indeed, regarding the specification, by using ATL, the procedure easily reduces to 2EXPTIME. This is due the fact that it suffices to build a Büchi PD-NTA of single exponential size [Alur et al., 2002]. Further, one can restrict to pushdown models with bounded-stack. In several settings, it has been shown that under such a restriction the problem has the same or slightly higher complexity than the corresponding one for finite-state systems [Alur and Yannakakis, 2001, Aminof et al., 2012]. By employing techniques similar to the ones reported in [Alur and Yannakakis, 2001], we are confident that the model-checking problem of ATL specifications over bounded-stack PGS is PTIME-COMPLETE as it is for the case for CGS. If so, one can think of implementing an efficient model checker, as it has been done with **MCMAS** [Lomuscio and Raimondi, 2006, Cermák et al., 2014]. This will be addressed as future work.

Another interesting setting to investigate is that of *imperfect information* under memoryless strategies. We recall that this setting is decidable in the finite-state case [Alur et al., 2002]. However, moving to pushdown systems one has to distinguish whether the missing information relies in the locations, in the pushdown store, or both. We recall that in pushdown module checking only the former case is decidable for specification given in CTL and CTL* [Aminof et al., 2007, Aminof et al., 2013].

References

[Abdulla et al.,] Abdulla, P. A., Atig, M. F., Hofman, P., Mayr, R., Kumar, K. N., and Totzke, P. Infinite-state energy games. In *CSL-LICS'14*.

[Alur et al., 2005] Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T. W., and Yannakakis, M. (2005). Analysis of Recursive State Machines. *ACM Trans. Program. Lang. Syst.*, 27(4):786–818.

[Alur et al., 2002] Alur, R., Henzinger, T., and Kupferman, O. (2002). Alternating-Time Temporal Logic. *JACM*, 49(5):672–713.

[Alur and Yannakakis, 2001] Alur, R. and Yannakakis, M. (2001). Model Checking of Hierarchical State Machines. *ACM Trans. Program. Lang. Syst.*, 23(3):273–303.

[Aminof et al., 2012] Aminof, B., Kupferman, O., and Murano, A. (2012). Improved Model Checking of Hierarchical Systems. *Inf. Comput.*, 210:68–86.

[Aminof et al., 2013] Aminof, B., Legay, A., Murano, A., Serre, O., and Vardi, M. Y. (2013). Pushdown Module Checking with Imperfect Information. *Inf. Comput.*, 213:1–17.

[Aminof et al., 2007] Aminof, B., Murano, A., and Vardi, M. (2007). Pushdown Module Checking with Imperfect Information. In *CONCUR '07*, LNCS 4703, pages 461–476. Springer-Verlag.

[Bouajjani et al., 1997] Bouajjani, A., Esparza, J., and Maler, O. (1997). Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *CONCUR'97*, pages 135–150.

[Bozzelli et al., 2005] Bozzelli, L., Murano, A., and Peron, A. (2005). Pushdown Module Checking. In *LPAR'05*, LNCS 3835, pages 504–518. Springer-Verlag.

[Bozzelli et al., 2010] Bozzelli, L., Murano, A., and Peron, A. (2010). Pushdown Module Checking. *FMSD*, 36(1):65–95.

[Bulling, 2014] Bulling, N. (2014). A Survey of Multi-Agent Decision Making. *KI*, 28(3):147–158.

[Cermák et al., 2014] Cermák, P., Lomuscio, A., Mogavero, F., and Murano, A. (2014). MCMAS-SLK: A Model Checker for the Verification of Strategy Logic Specifications. In *CAV*, pages 525–532.

[Cermák et al., 2015] Cermák, P., Lomuscio, A., and Murano, A. (2015). Verifying and Synthesising Multi-Agent Systems against One-Goal Strategy Logic Specifications. In *AAAI'15*, pages 2038–2044.

[Chatterjee and Velner, 2012] Chatterjee, K. and Velner, Y. (2012). Mean-Payoff Pushdown Games. In *LICS'12*, pages 195–204.

[Clarke and Emerson, 1981] Clarke, E. and Emerson, E. (1981). Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *LP'81*, LNCS 131, pages 52–71. Springer.

[Clarke et al., 2002] Clarke, E., Grumberg, O., and Peled, D. (2002). *Model Checking.* MIT Press.

[Kupferman et al., 2000a] Kupferman, O., M., P., T., P. S., and Vardi, M. Y. (2000a). Open Systems in Reactive Environments: Control and Synthesis. In *CONCUR'00*, pages 92–107.

[Kupferman et al., 2002] Kupferman, O., Piterman, N., and Vardi, M. Y. (2002). Pushdown Specifications. In *LPAR'02*, pages 262–277.

[Kupferman et al., 2000b] Kupferman, O., Vardi, M., and Wolper, P. (2000b). An Automata Theoretic Approach to Branching-Time Model Checking. *JACM*, 47(2):312–360.

[Kupferman et al., 2001] Kupferman, O., Vardi, M., and Wolper, P. (2001). Module Checking. *IC*, 164(2):322–344.

[Lomuscio and Raimondi, 2006] Lomuscio, A. and Raimondi, F. (2006). MCMAS: A Model Checker for Multi-agent Systems. In *TACAS'06*, pages 450–454.

[Mogavero et al., 2014] Mogavero, F., Murano, A., Perelli, G., and Vardi, M. (2014). Reasoning About Strategies: On the Model-Checking Problem. volume 15. doi:10.1145/2631917.

[Muller and Schupp, 1985] Muller, D. E. and Schupp, P. E. (1985). The Theory of Ends, Pushdown Automata, and Second-Order Logic. *Theor. Comput. Sci.*, 37:51–75.

[Piterman and Vardi, 2004] Piterman, N. and Vardi, M. Y. (2004). Global Model-Checking of Infinite-State Systems. In *CAV'04*, pages 387–400.

[Pnueli, 1977] Pnueli, A. (1977). The Temporal Logic of Programs. In *FOCS'77*, pages 46–57. IEEE Computer Society.

[Queille and Sifakis, 1981] Queille, J. and Sifakis, J. (1981). Specification and Verification of Concurrent Programs in Cesar. In *SP'81*, LNCS 137, pages 337–351. Springer.

[Schewe, 2008] Schewe, S. (2008). ATL* Satisfiability is 2ExpTime-Complete. In *ICALP'08*, LNCS 5126, pages 373–385. Springer.

[Song and Touili, 2014] Song, F. and Touili, T. (2014). Efficient CTL model-checking for pushdown systems. *Theor. Comput. Sci.*, 549:127–145.

[Vester, 2014] Vester, S. (2014). Model-checking Quantitative Alternating-time Temporal Logic on One-counter Game Models. Technical report, ArXiv.

[Walukiewicz, 1996] Walukiewicz, I. (1996). Pushdown Processes: Games and Model Checking. In *CAV'96*, pages 62–74.

[Walukiewicz, 2000] Walukiewicz, I. (2000). Model Checking CTL Properties of Pushdown Systems. In *FSTTCS'00*, pages 127–138.