

On the Complexity of ATL and ATL* Module Checking

Laura Bozzelli Aniello Murano
University of Napoli “Federico II”, Napoli, Italy

Module checking has been introduced in late 1990s to verify open systems, i.e., systems whose behavior depends on the continuous interaction with the environment. Classically, module checking has been investigated with respect to specifications given as CTL and CTL* formulas. Recently, it has been shown that CTL (resp., CTL*) module checking offers a distinctly different perspective from the better-known problem of ATL (resp., ATL*) *model checking*. In particular, ATL (resp., ATL*) module checking strictly enhances the expressiveness of both CTL (resp., CTL*) module checking and ATL (resp. ATL*) model checking. In this paper, we provide asymptotically optimal bounds on the computational cost of module checking against ATL and ATL*, whose upper bounds are based on an automata-theoretic approach. We show that module-checking for ATL is EXPTIME-complete, which is the same complexity of module checking against CTL. On the other hand, ATL* module checking turns out to be 3EXPTIME-complete, hence exponentially harder than CTL* module checking.

1 Introduction

Model checking is a well-established formal-method technique to automatically check for global correctness of systems [10, 27]. In this verification method, the behavior of a system, formally described by a mathematical model, is checked against a behavioral constraint specified by a formula in a suitable temporal logic. Originally, model checking was introduced to analyze finite-state *closed systems* whose dynamic behavior is completely determined by their internal states and transitions. In this specific setting, system models are usually given as labeled-state transition-graphs equipped with some internal degree of nondeterminism (e.g., Kripke structures). An unwinding of the graph results in an infinite tree, properly called *computation tree*, that collects all the possible evolutions of the system. Model checking of a closed system amounts to check whether the computation tree satisfies the specification. Properties for model checking are usually specified in temporal logics such as LTL, CTL, and CTL* [26, 11], or alternating-time temporal logics such as ATL and ATL* [3], the latter ones being extensions of CTL and CTL*, respectively, which allow for reasoning about the strategic capabilities of groups of agents.

In the last two decades, interest has arisen in analyzing the behavior of individual components (or sets of components) in systems with multiple entities. The interest began in the field of *reactive systems*, which are characterized by a continuous interaction with their (external) environments. One of the first approaches introduced to model check finite-state reactive systems is *module checking* [18]. In this setting, the system is modeled as a *module* that interacts with its environment, and correctness means that a desired property must hold with respect to all possible interactions. Technically speaking, the module is a transition system whose states 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 module satisfies a property amounts to check that all such trees satisfy the property. This makes the module-checking problem harder to deal with. Indeed, while CTL (resp., CTL*) model checking is PTIME-complete (resp.,

PSPACE-complete) [11], CTL (resp., CTL^{*}) module checking is EXPTIME-complete (resp., 2EXPTIME-complete) [18] with a PTIME-complete complexity for a fixed-size formula.

For a long time, there has been a common belief that module checking of CTL/CTL^{*} is a special case of model checking of ATL/ATL^{*}. Because of that, active research on module checking subsided shortly after its conception. The belief has been recently refuted in [15]. There, it was proved that module checking includes two features inherently absent in the semantics of ATL/ATL^{*}, namely irrevocability and nondeterminism of strategies. This result has brought back the interests in module checking as an interesting formalism for the verification of open systems. In particular, in [15], several scenarios were discussed to show the usefulness of considering the features of both settings combined together. This has led to an extension of the module-checking framework to ATL/ATL^{*} specifications [15, 16]. Notably, it has been showed that ATL/ATL^{*} module checking is strictly more expressive than both CTL/CTL^{*} module checking and ATL/ATL^{*} model checking [15, 16]. The computational complexity aspects have been shortly discussed in [16], where it is claimed that the complexity of ATL/ATL^{*} module checking is not worse than that of CTL/CTL^{*} module checking.

In this paper, we demonstrate that the claim made in [16] is not correct for ATL^{*}. While ATL module checking has the same complexity as CTL module checking, ATL^{*} module checking turns out to be exponentially harder than CTL^{*} module checking, and precisely, 3EXPTIME-complete with a PTIME-complete complexity for a fixed-size formula¹. The upper bounds are obtained by applying an automata-theoretic approach. The matching lower bound for ATL^{*} is shown by a technically non-trivial reduction from the word problem for 2EXPSpace-bounded alternating Turing Machines.

Related work. Module checking was introduced in [18], and later extended in several directions. In [19], the basic CTL/CTL^{*} module-checking problem was extended to the setting where the environment has imperfect information about the state of the system. In [7], it was extended to infinite-state open systems by considering pushdown modules. The pushdown module-checking problem was first investigated for perfect information, and later, in [4, 6], for imperfect information; the latter variant was proved in general undecidable in [4]. [13] address module checking against μ -calculus specifications, and in [25], the module-checking problem was studied for bounded pushdown modules (or *hierarchical modules*). From a more practical point of view, [23] built a semi-automated tool for module checking against the existential fragment of CTL, both in the perfect and imperfect information setting. A tableaux-based approach to CTL module-checking was also exploited in [5]. Finally, an extension of module checking was used to reason about three-valued abstractions in [2, 14].

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 all $i, j \in \mathbb{N}$, with $i \leq j$, $[i, j]$ denotes the set of natural numbers h such that $i \leq h \leq j$. For an infinite word w over an alphabet Σ and $i \geq 0$, $w(i)$ denotes the i^{th} letter of w and $w_{\geq i}$ the suffix of w given by $w(i)w(i+1)\dots$.

Given a set Υ of directions, an (*infinite*) Υ -tree T is a prefix closed subset of Υ^* such that for all $v \in T$,

¹The incorrect claim in [16] was due to a misleading interpretation of the result due to Schewe regarding 2EXPTIME-completeness for the ATL^{*} satisfiability problem [29].

$v \cdot \gamma \in T$ for some $\gamma \in \Upsilon$. Elements of T are called nodes and ε is the root of T . For $v \in T$, the set of children of v in T is the set of nodes of the form $v \cdot \gamma$ for some $\gamma \in \Upsilon$. A 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 infinite paths in the obvious way, i.e. $Lab(\pi)$ denotes the infinite word over Σ given by $Lab(\pi(0))Lab(\pi(1)) \dots$. The labeled tree $\langle T, Lab \rangle$ is *complete* if $T = \Upsilon^*$.

2.1 Concurrent Game Structures

Concurrent game structures (CGS) [3] generalize labeled transition systems to a setting with multiple agents (or players). 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 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 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 $\tau : S \times Dc \mapsto S \cup \{\top\}$ is a transition function that maps a state and a full decision either to a state or to the special symbol \top (\top is for ‘undefined’) such that for all states s , there exists $d \in Dc$ so that $\tau(s, d) \in S$. The CGS \mathcal{G} is finite if S is finite. 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 $\tau(s, d_A \cup d_{Ag \setminus A}) \in S$. We denote by $Dc_A(s)$ the nonempty set of A -decisions available at state 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.

We now recall the notion of strategy and counter strategy in a CGS $\mathcal{G} = \langle S, s_0, Lab, \tau \rangle$. For a state s , the set of successors of s is the set of states s' such that $s' = \tau(s, d)$ for some full decision d . A play is an infinite sequence of states $s_1 s_2 \dots$ such that s_{i+1} is a successor of s_i for all $i \geq 1$. A path (or track) v is a nonempty prefix of some play. Let Trk be the set of paths in \mathcal{G} . Given a set $A \subseteq Ag$ of agents, a strategy for A is a mapping $f_A : Trk \mapsto Ac^A$ assigning to each path v an A -decision available at the last state, denoted $lst(v)$, of v . For a state s , the set $out(s, f_A)$ of plays consistent with f_A starting from state s is given by $\{s_1 s_2 \dots \mid s_1 = s \text{ and } \forall i \geq 1 \exists d \in Ac^{Ag \setminus A}. s_{i+1} = \tau(s_i, f_A(s_1 \dots s_i) \cup d)\}$.

A counter strategy f_A^c for A is a mapping assigning to each track v a function $f_A^c(v) : Dc_A(lst(v)) \mapsto Ac^{Ag \setminus A}$, where the latter assigns to each A -decision d_A available at $lst(v)$ an $(Ag \setminus A)$ -decision $d_{Ag \setminus A}$ such that $\tau(lst(v), d_A \cup d_{Ag \setminus A}) \in S$. For a state s , the set $out(s, f_A^c)$ of plays consistent with the counter strategy f_A^c starting from state s is given by:

$$\{s_1 s_2 \dots \mid s_1 = s \text{ and } \forall i \geq 1 \exists d \in Dc_A(s_i). s_{i+1} = \tau(s_i, d \cup [f_A^c(s_1 \dots s_i)](d))\}$$

Definition 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 set of successors of x corresponds to the set of 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 T is the set of elements v in S^* such that $s_0 \cdot v$ is a track of \mathcal{G} , and for all $v \in T$ and $d \in Dc$, $Lab'(v) = Lab(lst(v))$ and $\tau'(v, d) = \tau(lst(v), d)$, where $lst(\varepsilon) = s_0$.

2.2 Alternating-Time Temporal Logics ATL* and ATL

We recall the alternating-temporal logics ATL* and ATL proposed by Alur et al. [3] as extensions of the standard branching-time temporal logics CTL* and CTL [11], 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 by the following grammar:

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

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 shorthands: the universal strategic quantifier $\llbracket A \rrbracket\varphi := \neg\langle\langle A \rangle\rangle\neg\varphi$, expressing that no strategy of A can prevent property φ , the eventually temporal modality $F\varphi := \text{true} U \varphi$, and the always temporal modality $G\varphi := \neg F\neg\varphi$. 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):

$$\begin{aligned} \mathcal{G}, \pi \models p & \Leftrightarrow p \in Lab(\pi(0)), \\ \mathcal{G}, \pi \models X\varphi & \Leftrightarrow \mathcal{G}, \pi_{\geq 1} \models \varphi, \\ \mathcal{G}, \pi \models \varphi_1 U \varphi_2 & \Leftrightarrow \exists j \geq 0 : \mathcal{G}, \pi_{\geq j} \models \varphi_2 \text{ and } \mathcal{G}, \pi_{\geq k} \models \varphi_1 \text{ for all } k \in [0, j-1] \\ \mathcal{G}, \pi \models \langle\langle A \rangle\rangle\varphi & \Leftrightarrow \text{for some strategy } f_A \text{ for } A, \mathcal{G}, \pi' \models \varphi \text{ for all } \pi' \in out(\pi(0), f_A). \end{aligned}$$

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 $Unw(\mathcal{G}) \models \varphi$.

Remark 1. By [29], for a state formula of the form $\llbracket A \rrbracket\varphi$, $\mathcal{G}, s \models \llbracket A \rrbracket\varphi$ iff there is a counter strategy f_A^c for A such that for all $\pi \in out(s, f_A^c)$, $\mathcal{G}, \pi \not\models \varphi$.

2.3 ATL^* and ATL Module checking

Module checking was proposed in [18] for the verification of finite open systems, that is systems that interact with an environment whose behavior cannot be determined in advance. In such a framework, the system is modeled by a *module* corresponding to a two-player turn-based game between the system and the environment. Thus, in a module, the set of states is partitioned into a set of system states (controlled by the system) and a set of environment states (controlled by the environment). The module-checking problem takes two inputs: a module M and a branching-time temporal formula ψ . The idea is that the open system should satisfy the specification ψ no matter how the environment behaves. Let us consider the unwinding $Unw(M)$ of M into an infinite tree. Checking whether $Unw(M)$ satisfies ψ is the usual model-checking problem. On the other hand, for an open system, $Unw(M)$ describes the interaction of the system with a maximal environment, i.e. an environment that enables all the external nondeterministic choices. In order to take into account all the possible behaviors of the environment, we have to consider all the trees T obtained from $Unw(M)$ by pruning subtrees whose root is a successor of an environment state (pruning these subtrees correspond to disabling possible environment choices). Therefore, a module M satisfies ψ if all these trees T satisfy ψ . It has been recently proved [15] that module checking of CTL/CTL^* includes two features inherently absent in the semantics of ATL/ATL^* , namely irrevocability of strategies 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 multi-agent setting for handling specifications in ATL* [16]. We now recall this setting which turns out to be more expressive than both CTL* module checking and ATL* model checking [15, 16]. In this framework, one considers a generalization of modules, namely *open CGS* (called multi-agent modules in [16]).

Definition 3 (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 $v \in T$ and $d \in Dc$, $Lab'(v) = Lab(lst(v))$, and $\tau'(v, d) = \tau(lst(v), d)$ if $v \cdot \tau(lst(v), d) \in T$, and $\tau'(v, d) = \top$ otherwise, where $lst(\varepsilon) = s_0$. Moreover, for all $v \in T$, the following holds:

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

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. The (*finite*) *module-checking problem against* ATL (resp., ATL*) is checking for a given finite open CGS \mathcal{G} and an ATL formula (resp., ATL* state formula) φ whether $\mathcal{G} \models^r \varphi$.

3 Decision procedures

In this section, we provide an automata-theoretic framework for solving the module-checking problem against ATL and ATL*, which is based on the use of parity alternating automata for CGS (parity ACG) [30]. The proposed approach consists of two steps. For a finite CGS \mathcal{G} and an ATL formula (resp., ATL* state formula) φ , one first builds a parity ACG $\mathcal{A}_{\neg\varphi}$ accepting the set of CGT which satisfy $\neg\varphi$. Then $\mathcal{G} \models^r \varphi$ iff *no* strategy tree of \mathcal{G} is accepted by $\mathcal{A}_{\neg\varphi}$.

The rest of the section is organized as follows. In Subsection 3.1, we recall the framework of ACG and provide a translation of ATL* state formulas into equivalent parity ACG involving a double exponential blowup. For ATL, a linear-time translation into equivalent parity ACG of index 2 directly follows from [30]. Then, in Subsection 3.2, we show that given a finite CGS \mathcal{G} and a parity ACG \mathcal{A} , checking that no strategy tree of \mathcal{G} is accepted by ACG can be done in time singly exponential in the size of \mathcal{A} and polynomial in the size of \mathcal{G} .

3.1 From ATL* to parity ACG

First, we recall the class of parity ACG [30]. For a set X , $\mathbb{B}^+(X)$ denotes the set of *positive* Boolean formulas over X , i.e. Boolean formulas built from elements in X using \vee and \wedge .

A parity ACG over 2^{AP} and Ag is a tuple $\mathcal{A} = \langle Q, q_0, \delta, \alpha \rangle$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times 2^{AP} \rightarrow \mathbb{B}^+(Q \times \{\square, \diamond\} \times 2^{Ag})$ is the transition function, and $\alpha : Q \mapsto \mathbb{N}$ is a parity acceptance condition over Q assigning to each state a color. The transition function δ maps a state and an input letter to a positive Boolean combination of universal atoms (q, \square, A) which refer to *all* successor states for some available A -decision, and existential atoms (q, \diamond, A) which refer to *some* successor state for all available A -decisions. The *index* of \mathcal{A} is the number of colors in α , i.e., the cardinality of $\alpha(Q)$. The size $|\mathcal{A}|$ of \mathcal{A} is $|Q| + |\text{Atoms}(\mathcal{A})|$, where $\text{Atoms}(\mathcal{A})$ is the set of atoms of \mathcal{A} , i.e. the set of tuples in $Q \times \{\square, \diamond\} \times 2^{Ag}$ occurring in the transition function δ of \mathcal{A} .

We interpret the parity ACG \mathcal{A} over CGT. Given a CGT $\mathcal{T} = \langle T, \varepsilon, \text{Lab}, \tau \rangle$ over AP and Ag , a run of \mathcal{A} over \mathcal{T} is a $(Q \times T)$ -labeled \mathbb{N} -tree $r = \langle T_r, \text{Lab}_r \rangle$, where each node of T_r labelled by (q, v) describes a copy of the automaton that is in the state q and reads the node v 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 $r(y) = (q, v)$, there is a set $H \subseteq Q \times \{\square, \diamond\} \times 2^{Ag}$ such that H is model of $\delta(q, \text{Lab}(v))$ and the set L of labels associated with the children of y in T_r minimally satisfies the following conditions:

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

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 $\text{Lab}_r(\pi)$ 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} .

It is well-known that ATL^* satisfiability has the same complexity as CTL^* satisfiability, i.e., it is 2EXPTIME -complete [29]. In particular, given an ATL^* state formula φ , one can construct in singly exponential time a parity ACG accepting the set of CGT satisfying some special requirements (depending on φ) which provide a necessary and sufficient condition for ensuring the existence of some model of φ [29]. These requirements are based on an equivalent representation of the models of a formula obtained by a sort of widening operation. When applied to the strategy trees of a finite CGS, such an encoding is not regular since one has to require that for all nodes in the encoding which are copies of the same environment node in the given strategy tree, the associated subtrees are isomorphic. Hence, the approach exploited in [29] cannot be applied to the module-checking setting. Here, by adapting the construction in [29], we provide a double exponential-time translation of ATL^* state formulas into equivalent parity ACG. In particular, we establish the following result, where for a finite set B disjunct from AP and a CGT $\mathcal{T} = \langle T, \varepsilon, \text{Lab}, \tau \rangle$ over AP , a B -labeling extension of \mathcal{T} is a CGT over $AP \cup B$ of the form $\langle T, \varepsilon, \text{Lab}', \tau \rangle$, where $\text{Lab}'(v) \cap AP = \text{Lab}(v)$ for all $v \in T$.

Theorem 1. *For an ATL^* state formula Φ over AP , one can construct in doubly exponential time a parity ACG \mathcal{A}_Φ over $2^{AP \cup B_\Phi}$, 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}_Φ . Moreover, \mathcal{A}_Φ has size $O(2^{2^{O(|\Phi| \cdot \log(|\Phi|))}})$ and index $2^{O(|\Phi|)}$.*

We now illustrate the proof of Theorem 1. For an ATL^* formula φ over AP , a *first-level basic subformula* of φ is a basic subformula of φ for which there is an occurrence in φ which is not in the scope of any strategy quantifier. Note that an ATL^* formula φ can be seen as a standard LTL formula [26], denoted $[\varphi]_{\text{LTL}}$, over the set AP augmented with the set of first-level basic subformulas of φ . In particular, if φ is a state formula, then $[\varphi]_{\text{LTL}}$ is a propositional formula. Fix an ATL^* state formula Φ over AP , and let B_Φ be the set of basic subformulas of Φ . Given a basic subformula $\langle\langle A \rangle\rangle \psi \in B_\Phi$ and a CGT $\mathcal{T} = \langle T, \varepsilon, \text{Lab}_\Phi, \tau \rangle$ over $AP \cup B_\Phi$, \mathcal{T} is *positively* (resp., *negatively*) *well-formed with respect to* $\langle\langle A \rangle\rangle \psi$ if:

- for all nodes $v \in T$ such that $\langle\langle A \rangle\rangle\psi \in \text{Lab}_\Phi(v)$ (resp., $\langle\langle A \rangle\rangle\psi \notin \text{Lab}_\Phi(v)$), there exists a strategy f_A (resp., counter strategy f_A^c) in \mathcal{T} for the set A of agents such that for all plays π in \mathcal{T} starting from v which are consistent with f_A (resp., f_A^c), it holds that $\text{Lab}_\Phi(\pi)$ is a model of the LTL formula $[\psi]_{\text{LTL}}$ (resp., $[\neg\psi]_{\text{LTL}}$).

The CGT \mathcal{T} is *well-formed with respect to Φ* if: (i) for all basic subformulas $\langle\langle A \rangle\rangle\psi \in B_\Phi$, \mathcal{T} is both positively and negatively well-formed w.r.t. $\langle\langle A \rangle\rangle\psi \in B_\Phi$, and (ii) $\text{Lab}_\Phi(\varepsilon)$ is a model of the propositional formula $[\Phi]_{\text{LTL}}$. The following proposition easily follows from the semantics of ATL* and the remark at the end of Section 2.2.

Proposition 1. *Given a 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 well-formed w.r.t. Φ .*

We show the following result that together with Proposition 1 provides a proof of Theorem 1.

Theorem 2. *Given an ATL* state formula Φ , one can construct in time doubly exponential in the size of Φ , a parity ACG \mathcal{A}_Φ over $2^{AP \cup B_\Phi}$ accepting the set of CGT over $AP \cup B_\Phi$ which are well-formed w.r.t. Φ . Moreover, \mathcal{A}_Φ has size $O(2^{2^{O(|\Phi| \cdot \log(|\Phi|))}})$ and index $2^{O(|\Phi|)}$.*

In order to prove Theorem 2, we exploit the well-known translation of LTL into Büchi nondeterministic word automata (Büchi NWA) [31]. In particular, given an LTL formula ψ , one can construct in singly exponential time a Büchi NWA accepting the set of infinite words which are models of ψ [31]. In order to handle a basic subformula of the form $\langle\langle Ag \rangle\rangle\psi$ and its negation ($\langle\langle Ag \rangle\rangle$ and $\neg\langle\langle Ag \rangle\rangle$ correspond to the existential and universal path quantifiers of CTL*), it suffices to use the Büchi NWA \mathcal{A}_ψ associated with $[\psi]_{\text{LTL}}$ and the dual $\tilde{\mathcal{A}}_\psi$ of \mathcal{A}_ψ , respectively ($\tilde{\mathcal{A}}_\psi$ is a universal co-Büchi word automaton). Indeed, for checking that $\langle\langle Ag \rangle\rangle\psi$ holds at the current node v of the input, the ACG simply guesses an infinite path π from v and simulates a run of \mathcal{A}_ψ over the labeling of π , and checks that it is accepting by using its parity acceptance condition. Similarly, for the formula $\neg\langle\langle Ag \rangle\rangle\psi$, the ACG simulates the universal co-Büchi word automaton $\tilde{\mathcal{A}}_\psi$ for checking that all the plays starting from v satisfy the LTL formula $[\neg\psi]_{\text{LTL}}$. This reasoning is the key for translating CTL* formulas into equivalent parity alternating tree automata with a single exponential blowup [20]. However, for handling more general basic subformulas $\langle\langle A \rangle\rangle\psi$ and their negations, we need to use deterministic word automata for the LTL formulas $[\psi]_{\text{LTL}}$ and $[\neg\psi]_{\text{LTL}}$. This because the choices of an ACG are local, and the set of plays starting from the current input node which are consistent with a strategy (resp., counter strategy) of A may be infinite and properly contained in the set of all the plays starting from v . The determinization of a Büchi NWA involves an additional exponential blowup [28].

3.2 Upper bounds for ATL and ATL* module checking

In this section, we establish the following result.

Theorem 3. *Given a CGS \mathcal{G} over AP , a finite set B disjoint from AP , and a parity ACG \mathcal{A} over $2^{AP \cup B}$, checking whether there are no B -labeling extensions of strategy trees of \mathcal{G} accepted by \mathcal{A} can be done in time singly exponential in the size of \mathcal{A} and polynomial in the size of \mathcal{G} .*

By [29], ATL can be translated in linear time into equivalent parity ACG of index 2. Thus, by Theorem 1 and Theorem 3, and since the CTL module-checking problem is EXPTIME-complete, and PTIME-complete for a fixed CTL formula, we obtain the following corollary.

Corollary 1. *The ATL* module-checking problem is in 3EXPTIME while the ATL module-checking problem is EXPTIME-complete. Moreover, for a fixed ATL* state formula (resp., ATL formula), the module-checking problem is PTIME-complete.*

In Section 4, we provide a lower bound for the ATL^* module-checking problem matching the upper bound in the corollary above. We now illustrate the proof of Theorem 3. We assume that the set B in the statement of Theorem 3 is empty (the general case where $B \neq \emptyset$ is similar). Let $\mathcal{G} = \langle S, s_0, Lab, \tau \rangle$ be a finite CGS over AP . Note that the transition function τ' of a strategy tree $\mathcal{T} = \langle T, \varepsilon, Lab', \tau' \rangle$ of \mathcal{G} is completely determined by T and the transition function τ of \mathcal{G} . Hence, for the fixed CGS \mathcal{G} , \mathcal{T} can be simply specified by the underlying 2^{AP} -labeled tree $\langle T, Lab' \rangle$. We consider an equivalent representation of $\langle T, Lab' \rangle$ by the $(2^{AP} \cup \{\perp\})$ -labeled *complete* S -tree $\langle S^*, Lab_{\perp} \rangle$, called the \perp -completion encoding of \mathcal{T} (\perp is a fresh proposition used to denote “completion” nodes), defined as: for each concrete node $v \in T$, $Lab_{\perp}(v) = Lab'(v)$, while for each completion node $v \in S^* \setminus T$, $Lab_{\perp}(v) = \{\perp\}$.

By the semantics of ACG, given a parity ACG \mathcal{A} with n states and index k , we can easily construct in polynomial time a standard parity alternating tree automaton (ATA) \mathcal{A}_g over the alphabet $S \times (2^{AP} \cup \{\perp\})$ and the set S of directions, having $O(n)$ -states and index k , accepting the set of $S \times (2^{AP} \cup \{\perp\})$ -labeled complete S -trees $\langle S^*, Lab \rangle$ such that for each $v \in T$, the S -label of v coincides with the direction $lst(v)$, and the labeled tree obtained from $\langle S^*, Lab \rangle$ by removing the S -labeling component is the \perp -completion encoding of a strategy tree of \mathcal{G} accepted by \mathcal{A} . However, this approach has an inconvenient. Indeed, in order to check emptiness of the parity ATA \mathcal{A}_g , one first constructs an equivalent parity nondeterministic tree automaton (NTA) \mathcal{A}'_g , and then checks for emptiness of \mathcal{A}'_g . By [12, 32], \mathcal{A}'_g has index polynomial in the size of the ACG \mathcal{A} , and number of states which is singly exponential both in the size of \mathcal{A} and in the number of directions, which in our case, coincides with the number of \mathcal{G} -states. We show that due to the form of the transition function of an ACG (it is independent of the set of directions), the exponential blowup in the number of \mathcal{G} -states can be avoided. In particular, by adapting the construction provided in [32] for converting parity two-way ATA into equivalent parity NTA, we provide a direct translation into parity NTA as established in the following Theorem 4. Since nonemptiness of parity NTA with n states and index k can be solved in time $O(n^k)$ [17], by Theorem 4, Theorem 3 (for the case $B = \emptyset$) directly follows.

Theorem 4. *Given a finite CGS $\mathcal{G} = \langle S, s_0, Lab, \tau \rangle$ over AP and an ACG $\mathcal{A} = \langle Q, q_0, \delta, \alpha \rangle$ over 2^{AP} with index k , one can construct in singly exponential time, a parity NTA \mathcal{A}_g over $2^{AP} \cup \{\perp\}$ and the set S of directions such that \mathcal{A}_g accepts the set of $2^{AP} \cup \{\perp\}$ -labeled complete S -trees which are the \perp -completion encodings of the strategy trees of \mathcal{G} which are accepted by \mathcal{A} . Moreover, \mathcal{A}_g has index $O(k^{|\mathcal{A}|^2})$ and $O(S \cdot (k^{|\mathcal{A}|^2})^{O(k^{|\mathcal{A}|^2})})$ states.*

4 3EXPTIME–hardness of ATL^* module checking

In this section, we establish the following result.

Theorem 5. *Module checking against ATL^* is 3EXPTIME–hard even for two-player turn-based open CGS of fixed size.*

Theorem 5 is proved by a polynomial-time reduction from the word problem for 2EXPSPACE–bounded alternating Turing Machines. Formally, an alternating Turing Machine (TM, for short) is a tuple $\mathcal{M} = \langle \Sigma, Q, Q_{\forall}, Q_{\exists}, q_0, \delta, F \rangle$, where Σ is the input alphabet, which contains the blank symbol $\#$, Q is the finite set of states which is partitioned into $Q = Q_{\forall} \cup Q_{\exists}$, Q_{\exists} (resp., Q_{\forall}) is the set of existential (resp., universal) states, q_0 is the initial state, $F \subseteq Q$ is the set of accepting states, and the transition function δ is a mapping $\delta : Q \times \Sigma \rightarrow (Q \times \Sigma \times \{L, R\})^2$. 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', dir) in $\delta(q, \sigma) = \langle (q_l, \sigma_l, dir_l), (q_r, \sigma_r, dir_r) \rangle$, and then

moves to state q' , writes σ' in the current tape cell, and its reading head moves one cell to the left or to the right, according to dir . 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 $\langle\langle q_l, \sigma_l, dir_l \rangle, \langle q_r, \sigma_r, dir_r \rangle\rangle$. 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 corresponds to 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 iff every leaf is labeled by an accepting configuration. An input $\alpha \in \Sigma^*$ is *accepted* by \mathcal{M} iff there is an accepting computation tree of \mathcal{M} over α . If \mathcal{M} is 2EXPSPACE-bounded, then there is a constant $k \geq 1$ such that for each $\alpha \in \Sigma^*$, the space needed by \mathcal{M} on input α is bounded by $2^{2^{|\alpha|^k}}$. It is well-known [8] that 3EXPTIME coincides with the class of all languages accepted by 2EXPSPACE-bounded alternating Turing Machines (TM). Moreover, the considered word problem remains 3EXPTIME-complete even for 2EXPSPACE-bounded TM of fixed size.

Fix a 2EXPSPACE-bounded TM $\mathcal{M} = \langle \Sigma, Q, Q_\forall, Q_\exists, q_0, \delta, F \rangle$ and an input $\alpha \in \Sigma^*$. Let $n = |\alpha|$. W.l.o.g. we assume that the constant k is 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 exactly 2^{2^n} . In particular, the initial configuration is $(q_0, \alpha(0))\alpha(1) \dots \alpha(n-1) \cdot (\#)^{2^n - n}$. Note that for a TM configuration $C = u_1 u_2 \dots u_{2^{2^n}}$ and for all $i \in [1, 2^{2^n}]$ 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 $i = 2^{2^n}$ and u_{i-1} for $i = 1$ to be some special symbol, say \perp). 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 above 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 $u_p, u_s \in \Sigma \cup (Q \times \Sigma) \cup \{\perp\}$.

In the following, we prove the following result from which Theorem 5 directly follows.

Theorem 6. *One can construct, in time polynomial in n and the size of \mathcal{M} , a finite turn-based open CGS \mathcal{G} 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 $exec(\mathcal{G})$ that satisfies φ iff $\mathcal{G} \not\models_r \neg\varphi$. Moreover, the size of \mathcal{G} depends only on the size of \mathcal{M} .*

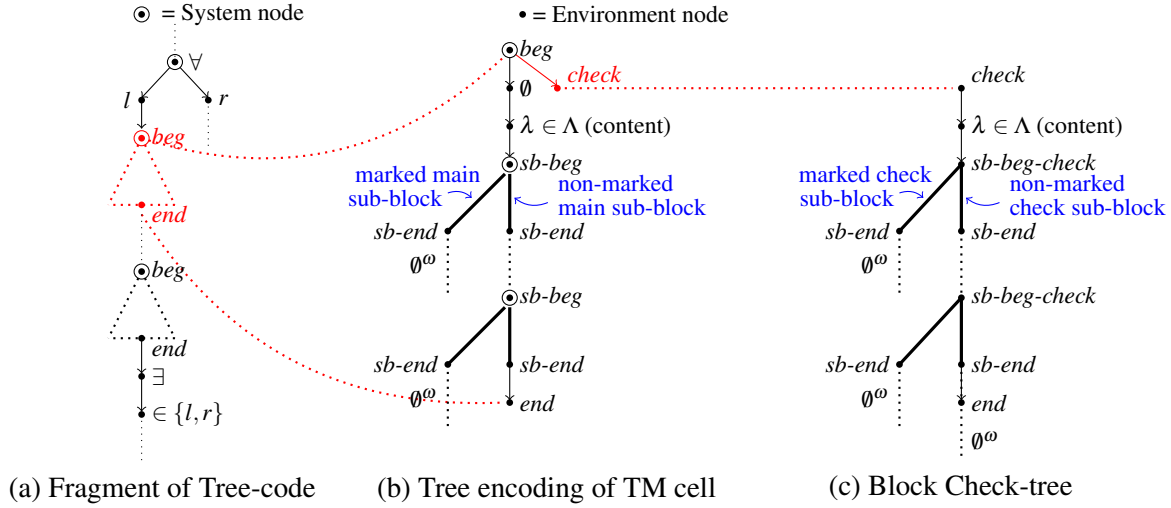
In order to prove Theorem 6, we first define a suitable encoding of the accepting computation trees of \mathcal{M} over α .

Encoding of computation trees of \mathcal{M} over α . In the encoding of a TM configuration, as usual, for each TM cell, we record both the content of the cell and the location (cell number) of the cell on the TM tape. We also record the contents of the previous and next cell (if any). Since the cell number is in the range $[0, 2^{2^n} - 1]$, it can be encoded by a 2^n -bit counter. Moreover, we need an n -bit counter in order to keep track of the *position* (index) of each bit of our 2^n -bit counter. Formally, we exploit the following set AP of atomic propositions

$$AP := \Lambda \cup \{0, 1, \forall, \exists, l, r, f, beg, end, check, sb-beg, sb-end, sb-beg-check, sb-mark\}$$

where 0 and 1 are used to encode the cell numbers, and the meaning of the letters in $\{\forall, \exists, l, r, f, beg, end, check, sb-beg, sb-end, sb-beg-check, sb-mark\}$ will be explained later.

The value $b \in \{0, 1\}$ and the index $i \in [0, 2^n - 1]$ of a bit in the 2^n -bit counter is encoded by a TM sub-block sb , which is a word of the form $sb = Type \cdot tag \cdot \{b\} \cdot \{b_1\} \cdot \dots \cdot \{b_n\} \cdot \{sb-end\}$, where $Type \in \{\{sb-beg\}, \{sb-beg-check\}\}$, $tag \in \{\emptyset, \{sb-mark\}\}$, and $b_1 \cdot \dots \cdot b_n \in \{0, 1\}^n$ is the binary code of the index i . We say that b (resp., i) is the *content* (resp., *number*) of sb . Moreover, sb is a *main* (resp.,

Figure 1: Encoding of computation trees of \mathcal{M} 

check) sub-block if $beg = \{sb-beg\}$ (resp., $beg = \{sb-beg-check\}$), and *sb* is *marked* (resp., *non-marked*) if $tag = \{sb-mark\}$ (resp., $tag = \emptyset$).

A TM cell is in turn encoded by a *TM block*, which is a word bl of the form $bl = \{beg\} \cdot tag \cdot \lambda \cdot sb_1 \cdot \dots \cdot sb_k \cdot \{end\}$ for some $k \geq 1$, where $tag \in \{\emptyset, \{check\}\}$, $\lambda \in \Lambda$ is the *content* of bl , and sb_1, \dots, sb_k are *non-marked main* sub-blocks if $tag = \emptyset$ (in this case, bl is a *main* block), and sb_1, \dots, sb_k are *non-marked check* sub-blocks otherwise (in this case, bl is a *check* block). If $k = 2^n$ and for each $i \in [1, 2^n]$, the number of sb_i is $i - 1$, we say that bl is *well-formed*. In this case, the *number* of bl is the integer in $[0, 2^{2^n} - 1]$ whose binary code is given by $b_1 \dots b_{2^n}$, where for all $i \in [1, 2^n]$, b_i is the content of sb_i . Note that if the content λ of bl is of the form (u_p, u, u_s) , then u represents the value of the encoded TM cell, while u_p (resp., u_s) represents the value of the previous (resp., next) cell in the TM configuration.

TM configurations $C = u_1 u_2 \dots u_k$ (note that here we do not require that $k = 2^{2^n}$) are then encoded by words w_C of the form $w_C = tag_1 \cdot bl_1 \cdot \dots \cdot bl_k \cdot tag_2$, where $tag_1 \in \{\{l\}, \{r\}\}$, for each $i \in [1, k]$, bl_i is a *non-marked main* TM block whose content is (u_{i-1}, u_i, u_{i+1}) (where $u_0 = \perp$ and $u_{k+1} = \perp$), $tag_2 = \{f\}$ 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 TM successor, respectively. We also use the symbol l to mark the initial configuration. If $k = 2^{2^n}$ and for each $i \in [1, k]$, bl_i is a well-formed block having number $i - 1$, then we say that w_C is a *well-formed code* of C . A sequence $w_{C_1} \dots w_{C_p}$ of well-formed TM 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 $C_{i+1} = succ_r(C_i)$.

In the encoding of the computation trees of \mathcal{M} , *marked* sub-blocks are used as additional branches for ensuring by a CTL* formula that the TM blocks are well-formed (i.e., the n -counter is properly updated) and the TM configurations codes are well-formed as well (i.e., the 2^n -counter is properly updated). Moreover, suitable tree encodings of *check* TM blocks, called *block check-trees* (see Figure 1(c)) are exploited as additional subtrees for ensuring by an ATL* formula that the encoding is faithful to the evolution of \mathcal{M} . Intuitively, a block check-tree corresponds to a *check* TM block bl extended with additional branches which represent marked copies of the sub-blocks of bl .

Definition 4 (Block Check-trees). A block check-tree is a 2^{AP} -labeled tree $\langle T, Lab \rangle$ such that there is an infinite path π from the root so that $Lab(\pi)$ is of the form $bl \cdot \emptyset^\omega$, where bl is a *check* block (bl is the block encoded by $\langle T, Lab \rangle$), and the following holds:

- each node x of π labeled by $\{sb\text{-beg-check}\}$ (the first symbol of a sub-block of bl) has two children, and for the child y of x which is not visited by π , there is a unique infinite path π' from x and visiting y . Moreover, $Lab(\pi')$ is of the form $sb \cdot \emptyset^\omega$, where sb is a marked check sub-block (sb is the companion of the main sub-block of π associated with node x);
- each node of π which is not labeled by $\{sb\text{-beg-check}\}$ has exactly one child.

$\langle T, Lab \rangle$ is well-formed if, additionally, $Lab(\pi)$ encodes a well-formed check block and for each sub-block sb along π , the companion sb' of sb has the same content and number as sb .

We now define an encoding of the computation trees of \mathcal{M} (see Figure 1), where, intuitively, the computations paths (main paths) are extended with additional branches (marked main sub-blocks) and additional subtrees (block check-trees).

Definition 5 (Tree-Codes). A tree-code is a 2^{AP} -labeled tree $\langle T, Lab \rangle$ such that there is a set Π of infinite paths from the root, called main paths, so that for each $\pi \in \Pi$, $Lab(\pi) = w_\pi \cdot \emptyset^\omega$ where w_π is a sequence of codes of TM configurations C_1, \dots, C_p , C_1 has the form $(q_0, \alpha(0))\alpha(1) \dots \alpha(n-1) \cdot (\#)^k$ for some $k \geq 0$, C_p is accepting, C_i is not accepting for all $i \in [1, p-1]$, and the following holds for each node x along π :

- if x has label $\{\forall\}$, then x has two children, with labels $\{l\}$ and $\{r\}$, respectively, and for the child y of x which is not visited by π , there is a main path visiting y ;
- if x has label $\{sb\text{-beg}\}$, then x has two children, and for the child y of x which is not visited by π , there is a unique infinite path π' starting from x and visiting y . Moreover, $Lab(\pi')$ is of the form $sb \cdot \emptyset^\omega$, where sb is a marked main sub-block (sb is the companion of the non-marked main sub-block along π associated with node x);
- if x has label $\{beg\}$, then x has two children, and if we remove the child of x visited by π and all its descendants, then the resulting subtree rooted at node x is a block check-tree;
- if the label of x is not in $\{\{\forall\}, \{beg\}, \{sb\text{-beg}\}\}$, then x has exactly one child.

A tree-code $\langle T, Lab \rangle$ is well-formed if for each main path π , the following additionally holds:

- (i) TM configuration codes along w_π are well-formed, (ii) for each sub-block sb along π , the companion of sb has the same content and number as sb , and (iii) for each block bl along π , the associated block check-tree is well-formed and encodes a check block having the same number and content as bl .

A tree-code is fair, if for each main path π , w_π is faithful to the evolution of \mathcal{M} . Evidently, there is a fair well-formed tree-code iff there is an accepting computation tree of \mathcal{M} over α .

Construction of the open CGS \mathcal{G} and the ATL* formula φ in Theorem 6. By the definition of tree-codes, the following result (Lemma 1), concerning the construction of the open CGS in Theorem 6, trivially follows, where a minimal 2^{AP} -labeled tree is a 2^{AP} -labeled tree $\langle T, Lab \rangle$ whose root has label $\{l\}$ and satisfying the following:

- (i) for each node x , the children of x have distinct labels and $Lab(x)$ is either empty or a singleton;
- (ii) each node labeled by $\{sb\text{-beg}\}$ (resp., $\{beg\}$) has two children, one with empty label and the other one with label $\{sb\text{-mark}\}$ (resp., $\{check\}$); and (iii) each node labeled by $\{\forall\}$ has two children, with labels $\{l\}$ and $\{r\}$, respectively.

Lemma 1. One can construct in time polynomial in $|AP|$, a finite turn-based open CGS \mathcal{G} over AP and $Ag = \{env, sys\}$ satisfying the following:

- $Unw(\mathcal{G}) = \langle T, Lab, \tau \rangle$, where $\langle T, Lab \rangle$ is a minimal 2^{AP} -labeled tree;
- for each tree-code $\langle T', Lab' \rangle$, there is a strategy tree in $exec(\mathcal{G})$ of the form $\langle T', Lab', \tau' \rangle$;
- each state which is labeled by either $\{beg\}$ or $\{sb\text{-beg}\}$ or $\{\forall\}$ is controlled by the system;
- each state whose label is not in $\{\{beg\}, \{sb\text{-beg}\}, \{\forall\}\}$ is controlled by the environment.

According to Lemma 1, a minimal 2^{AP} -labeled tree can be interpreted as a two-player turn-based CGT between the environment and the system, where the nodes having label in $\{\{beg\}, \{beg\}, \{\forall\}\}$ are controlled by the system, while all the other nodes are controlled by the environment. With this interpretation, we now establish the following result that together with Lemma 1 provide a proof of Theorem 6.

Lemma 2. *One can construct in time polynomial in n and $|AP|$, an ATL* state formula φ over AP and $Ag = \{env, sys\}$ such that for each minimal 2^{AP} -labeled tree $\langle T, Lab \rangle$, $\langle T, Lab \rangle$ is a model of φ iff $\langle T, Lab \rangle$ is a fair well-formed tree-code.*

Proof. The ATL* formula φ is given by $\varphi := \varphi_{TC} \wedge \varphi_{WTC} \wedge \varphi_{fair}$, where: (i) φ_{TC} is a CTL* formula which is satisfied by a minimal 2^{AP} -labeled tree $\langle T, Lab \rangle$ iff $\langle T, Lab \rangle$ is a tree-code, (ii) φ_{WTC} is a CTL* formula requiring that each tree-code is well-formed, and (iii) φ_{fair} is an ATL* formula ensuring that a well-formed tree-code is fair. Here, we focus on the construction of the ATL* formula φ_{fair} . Let $\langle T, Lab \rangle$ be a well-formed tree-code, π be a main path of $\langle T, Lab \rangle$, and w_C be a non-terminal well-formed configuration code along π associated with a TM configuration C . Assume that the last symbol of w_C is \forall , i.e., C is universal (the other case, where the last symbol is \exists being similar). Let x be the node associated with the last symbol of w_C . Then, there are two configuration codes w_{C_l} and w_{C_r} associated with configurations C_l and C_r , respectively, such that the first symbol of w_{C_l} (resp., w_{C_r}) is $\{l\}$ (resp., $\{r\}$). Moreover, one of the codes follows w_C along π , while the other one follows w_C along a main path which visits the child of node x which is not visited by π . We have to require that for all $dir \in \{l, r\}$, $C_{dir} = succ_{dir}(C)$. This reduces to check that for each block bl of w_C , denoted by bl_{dir} the block of $w_{C_{dir}}$ having the same number as bl , and by (u_p, u, u_s) (resp., (u'_p, u', u'_s)) the content of block bl (resp., bl_{dir}), the following holds: $u' = next_{dir}(u_p, u, u_s)$. For this check, we exploit the block check-tree, say BCT , associated with the main block bl_{dir} , whose encoded check TM block (the companion of bl_{dir}) has the same content and number as bl_{dir} . Recall that all the nodes in BCT but the root (which is a $\{beg\}$ -labeled node) are controlled by the environment. Moreover, the unique nodes in $\langle T, Lab \rangle$ controlled by the system are the ones having label in $\{\{\forall\}, \{beg\}, \{sb-beg\}\}$. Let x_{bl} be the starting node for the selected block bl of w_C . Then, there is a strategy f_{bl} of the player system such that

- (i) each play consistent with the strategy f_{bl} starting from node x_{bl} gets trapped in the check-tree BCT , and (ii) each infinite path starting from node x_{bl} and leading to some marked sub-block of BCT is consistent with the strategy f_{bl} .

Note that each strategy of the system selects exactly one child for each node controlled by the system. Thus, the ATL* formula φ_{fair} “guesses” the strategy f_{bl} and ensures that the guess is correct by verifying the following conditions on the outcomes of f_{bl} from node x_{bl} :

1. each outcome visits a $\{check\}$ -node whose parent belongs to a block of $w_{C_{dir}}$. This ensures that all the outcomes get trapped in the *same* block check-tree associated with some block of $w_{C_{dir}}$. Moreover, for the label (u'_p, u', u'_s) of the node following the $\{check\}$ -node along the outcome, $u' = next_{dir}(u_p, u, u_s)$, where (u_p, u, u_s) is the content of bl .
2. for each outcome π' which leads to a marked sub-block sb' (note that this sub-block is necessarily in BCT), denoting by sb the sub-block of bl having the same number as sb , it holds that sb and sb' have the same content.

The first (resp., second) condition is implemented by the LTL formula ψ_{dir} (resp., ψ_{cor}) in the definition of φ_{fair} below.

$$\varphi_{fair} := \bigwedge_{dir \in \{l, r\}} \text{AG} \left((beg \wedge \text{EF } dir) \longrightarrow \langle\langle sys \rangle\rangle (\psi_{dir} \wedge \psi_{cor}) \right)$$

$$\begin{aligned} \Psi_{dir} &:= \bigvee_{(u_p, u, u_s), (u'_p, u', u'_s) \in \Lambda: u' = \text{next}_{dir}(u_p, u, u_s)} \left(X^2(u_p, u, u_s) \wedge \right. \\ &\quad \left. \left[(-l \wedge \neg r) \cup (dir \wedge X((-l \wedge \neg r) \cup (check \wedge X(u'_p, u', u'_s)))) \right] \right) \\ \Psi_{cor} &:= Fsb\text{-mark} \rightarrow \left((\neg end \wedge (sb\text{-beg} \rightarrow X\theta_{cor})) \cup end \right) \\ \theta_{cor} &:= \left(\bigwedge_{i=1}^{i=n} \bigvee_{b \in \{0,1\}} ((X^{i+1} b) \wedge F(sb\text{-mark} \wedge X^{i+1} b)) \right) \longrightarrow \bigvee_{b \in \{0,1\}} ((Xb) \wedge F(sb\text{-mark} \wedge Xb)) \end{aligned}$$

This concludes the proof of Lemma 2. □

5 Conclusion

Module checking is a useful game-theoretic framework to deal with branching-time specifications. The setting is simple and powerful as it allows to capture the essence of the adversarial interaction between an open system (possibly consisting of several independent components) and its unpredictable environment. The work on module checking has brought an important contribution to the strategic reasoning field, both in computer science and AI [3]. Recently, CTL/CTL* module checking has come to the fore as it has been shown that it is incomparable with ATL/ATL* model checking [15]. In particular the former can keep track of all moves made in the past, while the latter cannot. This is a severe limitation in ATL/ATL* and has been studied under the name of irrevocability of strategies in [1]. Remarkably, this feature can be handled with more sophisticated logics such as *Strategy Logics* [9, 24], *ATL with strategy contexts* [22], and *quantified CTL* [21]. However, for such logics, the relative model checking question turns out to be non-elementary.

In this paper, we have addressed and carefully investigated the computational complexity of the module-checking problem against ATL and ATL* specifications. We have shown that ATL module-checking is EXPTIME-complete, while ATL* module-checking is 3EXPTIME-complete. The latter corrects an incorrect claim made in [16]. Note that following [22], ATL* (resp., ATL) module-checking can be reduced to model checking against *quantified CTL** (resp., *quantified CTL*), but this approach would lead to non-elementary algorithms for the considered problems. This work opens to several directions for future work. Mainly, we aim to investigate the same problem in the imperfect information setting as well as for infinite-state open systems.

References

- [1] T. Ågotnes, V. Goranko & W. Jamroga (2007): *Alternating-time temporal logics with irrevocable strategies*. In: *TARK'07*, pp. 15–24, doi:10.1145/1324249.1324256.
- [2] L. de Alfaro, P. Godefroid & R. Jagadeesan (2004): *Three-Valued Abstractions of Games: Uncertainty, but with Precision*. In: *LICS'04*, IEEE, pp. 170–179, doi:10.1109/LICS.2004.1319611.
- [3] R. Alur, T. A. Henzinger & O. Kupferman (2002): *Alternating-time temporal logic*. *Journal of the ACM* 49(5), pp. 672–713, doi:10.1145/585265.585270.
- [4] B. Aminof, A. Legay, A. Murano, O. Serre & M. Y. Vardi (2013): *Pushdown module checking with imperfect information*. *Inf. Comput.* 223(1), pp. 1–17, doi:10.1016/j.ic.2012.11.005.
- [5] S. Basu, P. S. Roop & R. Sinha (2007): *Local Module Checking for CTL Specifications*. *ENTCS* 176 (2), pp. 125–141, doi:10.1016/j.entcs.2006.02.035.
- [6] L. Bozzelli (2011): *New results on pushdown module checking with imperfect information*. In: *GandALF'11*, EPTCS 54, pp. 162–177, doi:10.4204/EPTCS.54.12.

- [7] L. Bozzelli, A. Murano & A. Peron (2010): *Pushdown Module Checking*. *Formal Methods in System Design* 36(1), pp. 65–95, doi:10.1007/s10703-010-0093-x.
- [8] A.K. Chandra, D.C. Kozen & L.J. Stockmeyer (1981): *Alternation*. *Journal of the ACM* 28(1), pp. 114–133, doi:10.1145/322234.322243.
- [9] K. Chatterjee, T. A. Henzinger & N. Piterman (2010): *Strategy logic*. *Inf. Comput.* 208(6), pp. 677–693, doi:10.1016/j.ic.2009.07.004.
- [10] E.M. Clarke & E.A. Emerson (1981): *Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic*. In: *LP'81*, LNCS 131, pp. 52–71, doi:10.1007/BFb0025774.
- [11] E.A. Emerson & J.Y. Halpern (1986): "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *Journal of the ACM* 33(1), pp. 151–178, doi:10.1145/4904.4999.
- [12] E.A. Emerson & C.S. Jutla (1988): *The Complexity of Tree Automata and Logics of Programs*. In: *FOCS'88*, pp. 328–337, doi:10.1109/SFCS.1988.21949.
- [13] A. Ferrante, A. Murano & M. Parente (2008): *Enriched μ -Calculi Module Checking*. *Logical Methods in Computer Science* 4(3:1), pp. 1–21, doi:10.2168/LMCS-4(3:1)2008.
- [14] P. Godefroid (2003): *Reasoning about Abstract Open Systems with Generalized Module Checking*. In: *EMSOFT'03*, LNCS 2855, Springer, pp. 223–240, doi:10.1007/978-3-540-45212-6_15.
- [15] W. Jamroga & A. Murano (2014): *On module checking and strategies*. In: *AAMAS'14*, IFAAMAS/ACM, pp. 701–708.
- [16] W. Jamroga & A. Murano (2015): *Module Checking of Strategic Ability*. In: *AAMAS'15*, ACM, pp. 227–235.
- [17] O. Kupferman & M. Y. Vardi (1998): *Weak Alternating Automata and Tree Automata Emptiness*. In: *STOC'98*, ACM, pp. 224–233, doi:10.1145/276698.276748.
- [18] O. Kupferman & M.Y. Vardi (1996): *Module Checking*. In: *CAV'96*, LNCS 1102, Springer, pp. 75–86, doi:10.1007/3-540-61474-5_59.
- [19] O. Kupferman & M.Y. Vardi (1997): *Module Checking Revisited*. In: *CAV'97*, LNCS 1254, Springer, pp. 36–47, doi:10.1007/3-540-63166-6_7.
- [20] O. Kupferman, M.Y. Vardi & P. Wolper (2000): *An Automata-Theoretic Approach to Branching-Time Model Checking*. *Journal of the ACM* 47(2), pp. 312–360, doi:10.1145/333979.333987.
- [21] F. Laroussinie & N. Markey (2014): *Quantified CTL: Expressiveness and Complexity*. *Logical Methods in Computer Science* 10(4), doi:10.2168/LMCS-10(4:17)2014.
- [22] F. Laroussinie & N. Markey (2015): *Augmenting ATL with strategy contexts*. *Inf. Comput.* 245, pp. 98–123, doi:10.1016/j.ic.2014.12.020.
- [23] F. Martinelli & I. Matteucci (2007): *An Approach for the Specification, Verification and Synthesis of Secure Systems*. *ENTCS 168*, pp. 29–43, doi:10.1016/j.entcs.2006.12.003.
- [24] F. Mogavero, A. Murano, G. Perelli & M. Y. Vardi (2014): *Reasoning About Strategies: On the Model-Checking Problem*. *ACM Trans. Comput. Log.* 15(4), pp. 34:1–34:47, doi:10.1145/2631917.
- [25] A. Murano, M. Napoli & M. Parente (2008): *Program Complexity in Hierarchical Module Checking*. In: *LPAR'08*, LNCS 5330, Springer, pp. 318–332, doi:10.1007/978-3-540-89439-1_23.
- [26] A. Pnueli (1977): *The Temporal Logic of Programs*. In: *FOCS'77*, IEEE, pp. 46–57, doi:10.1109/SFCS.1977.32.
- [27] J.P. Queille & J. Sifakis (1982): *Specification and verification of concurrent programs in Cesar*. In: *SP'82*, LNCS 137, Springer, pp. 337–351, doi:10.1007/3-540-11494-7_22.
- [28] S. Safra (1988): *On the Complexity of ω -Automata*. In: *FOCS'88*, IEEE, pp. 319–327, doi:10.1109/SFCS.1988.21948.
- [29] S. Schewe (2008): *ATL* Satisfiability Is 2EXPTIME-Complete*. In: *ICALP'08*, LNCS 5126, Springer, pp. 373–385, doi:10.1007/978-3-540-70583-3_31.

- [30] S. Schewe & B. Finkbeiner (2006): *Satisfiability and Finite Model Property for the Alternating-Time mu-Calculus*. In: *CSL'06*, LNCS 4207, Springer, pp. 591–605, doi:10.1007/11874683_39.
- [31] M. Y. Vardi & P. Wolper (1994): *Reasoning About Infinite Computations*. *Inf. Comput.* 115(1), pp. 1–37, doi:10.1006/inco.1994.1092.
- [32] M.Y. Vardi (1998): *Reasoning about the past with two-way automata*. In: *ICALP'98*, LNCS 1443, Springer, pp. 628–641, doi:10.1007/BFb0055090.