On Module Checking and Strategies

Wojciech Jamroga
Computer Science and Communication
and Interdisc. Centre on Security and Trust
University of Luxembourg, Luxembourg
wojtek.jamroga@uni.lu

Aniello Murano
Dipartimento di Ingegneria Elettrica e
Tecnologie dell'Informazione
Università degli Studi di Napoli Federico II, Italy
murano@na.infn.it

ABSTRACT

Two decision problems are very close in spirit: *module checking* of CTL/CTL* and *model checking* of ATL/ATL*. The latter appears to be a natural multi-agent extension of the former, and it is commonly believed that model checking of ATL(*) subsumes module checking of CTL(*) in a straightforward way. Perhaps because of that, the exact relationship between the two has never been formally established.

A more careful look at the known complexity results, however, makes one realize that the relationship is somewhat suspicious. In particular, module checking of CTL is EXPTIME-complete, while model checking of ATL is only P-complete. Thus, the (seemingly) less expressive framework yields significantly higher computational complexity than the (seemingly) more expressive one. This suggests that the relationship may not be as simple as believed. In this paper, we show that the difference is indeed fundamental. The way in which behavior of the environment is understood in module checking cannot be equivalently characterized in ATL(*). Conversely, if one wants to embed module checking in ATL(*) then its semantics must be extended with two essential features, namely nondeterministic strategies and long-term commitment to strategies.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*Modal logic*

General Terms

Theory, Verification

Keywords

Module checking, model checking, verification, temporal logic, reactive systems.

1. INTRODUCTION

In design and verification of formal systems, *model checking* is a well-established method to automatically check for global correctness of systems [21, 45]. In such a framework, in order to verify whether a system is correct with respect to a desired property, we describe its structure with a mathematical model, specify the property with a temporal logic formula, and check formally that the model satisfies the formula. The method has been first conceived for *closed systems* whose behavior is completely determined by their internal states and transitions. In this setting, models are often given as *Kripke structures* (labeled state transition graphs). CTL* or its sublogics CTL, LTL [24] are usually used for specification of properties. It is worth observing that closed models may include internal nondeterminism. Hence, an unwinding of a Kripke structure results in an infinite tree, formally called *computation tree*, that collects all possible evolutions of the system. Then, model checking of a closed system amounts to checking whether the tree is correct with respect to the specification.

Module checking. 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 systems that interact continually with their environments. In *module checking* [37], 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. The module can be seen as a Kripke structure with states partitioned into ones controlled by the system and by the environment. Notice that the environment represents an external additional source of nondeterminism, because at each state controlled by the environment the computation can continue with any subset of its possible successor states. In other words, while in model checking we have only one computation tree to check, in module checking we have an infinite number of trees to handle, one for each possible behavior of the environment.

This makes the module checking problem harder to deal with. Indeed, while CTL* model checking is PSPACE-complete, CTL* module checking is 2EXPTIME-complete. Moreover, CTL model checking is P-complete, whereas CTL module checking is EXPTIME-complete. Finally, module checking is exponentially harder even in terms of program complexity (i.e., in case we use a fixed-size formula) where we move from LogSpace-completeness for model checking CTL to P-completeness for module checking CTL [37, 47].

Alternating-time logic. Taking module checking as the starting point, researchers have looked for logics to reason about, and verify strategic behavior of agents in multi-agent systems [7, 8, 44, 48, 32, 19, 41]. Perhaps the most impor-

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

tant development in this field was *alternating-time temporal logic* (ATL* for short), introduced by Alur, Henzinger, and Kupferman [7, 8]. ATL* allows reasoning about strategies of agents with temporal goals. Formally, it is obtained as a generalization of CTL* in which the path quantifiers E (*"there exists a path"*) and A (*"for all paths"*) are replaced with *strategic modalities* of the form $\langle\langle A\rangle\rangle$ (*"A can collectively enforce that..."*), where A is a set of *agents* (a.k.a. *players*). Strategic modalities are used to express cooperation and competition among agents in order to achieve certain goals. In particular, they can express selective quantification over those paths that are the result of the infinite game between coalition A and the rest of agents.

Module checking vs. ATL*. Model checking of ATL* comes out as a natural multi-agent extension of CTL* module checking [8] and it is commonly believed that the latter can be embedded by the former in a straightforward way [8, 14, 52]. However, the relationship between the two has never been formally shown, which is rather remarkable given how relevant the topic is for verification of open and multi-agent systems. This lack of formal correspondence results is not without a reason, and the existing complexity results suggest potential misalignment. True, the complexity of the two problems in their most general variant match, i.e., both module checking of CTL* and model checking of ATL* are 2EXPTIME-complete. On the other hand, for the state-based fragments, we get that model checking of ATL is only P-complete while module checking of CTL is EXPTIME-complete. The (seemingly) less expressive framework[1] yields significantly higher computational complexity than the (seemingly) more expressive one! Thus, the relationship cannot be as simple as commonly believed. Of course, there could be many reasons for this pattern of complexity (perhaps a translation to from CTL to ATL* is needed for the embedding, or the optimal translation requires exponential blowup of the formula etc.). In this paper, we show that none of those is the case, and the difference is fundamental.

Contribution of this paper. In essence, we show that the way in which the behavior of the environment is understood in module checking cannot be equivalently characterized in ATL*. The main intricacy is that in module checking we need to handle nondeterministic choices of the environment that affect the computation tree by *pruning* the non-selected transitions. This is something that cannot be done in the standard semantics of ATL*. In order to embed module checking in ATL* one must extend its semantics with nondeterministic strategies and the possibility of long-term commitment to strategies. We prove it formally by a number of (non) expressivity theorems.

Related work. Module checking is an active area of research. Since its introduction in [35, 37], it has been extensively studied in several directions. In [36], the basic CTL/CTL* module checking question has been extended to the setting where the environment has *imperfect information* about the state of the system, showing that such a constraint does not affect the overall complexity of the problem. In [13] the module checking problem has been extended to infinite-state open systems, by considering pushdown modules. The *pushdown module checking* problem has been first investigated for perfect information, showing that it is exponentially harder than in the finite-state case. Then, in [10, 12], the problem has been investigated for imperfect information. In [10], it has been proved that it is in general undecidable, and that the undecidability relies on hiding information about the pushdown store. [26, 9] extended module checking to μ-calculus specifications, and showed it to be as complex as in the CTL case. Finally, in [43] the module checking problem has been investigated with respect to bounded pushdown modules (formally *hierarchical modules*), pointing out a rare case in which the program complexity of the model and module checking problems coincide.

>From a more practical point of view, Martinelli [39] built a semi-automated tool to perform the finite-state module checking problem, both in the perfect and imperfect setting, with respect to a specification given in the existential fragment of CTL (see also [40]). An approach to CTL module checking based on tableau has been exploited in [11]. Moreover, Godefroid and Huth used an extension of module checking to reason about three-valued abstractions [28]. We refer to [23, 27, 29] for the evolution of this idea.

Literature on model checking of alternating-time temporal logic is equally rich. The complexity of the problem has been studied in a multitude of papers [8, 48, 30, 50, 17], cf. also [15] for an overview. Symbolic model checking algorithms for ATL were developed e.g. in [33, 46]. Existing implementations of model checkers include **mocha** [5, 4] and **MCMAS** [38], the latter constantly developed since 2004. Also, the probabilistic model checker **PRISM** has been recently extended to handle a probabilistic variant of ATL [20]. ATL has been studied as a framework for specification and verification in several domains, including communication protocols [51], fair exchange protocols [34, 31], and agent-oriented programs [22].

An important strand in research on ATL/ATL* emerged in quest of the "right" semantics for strategic ability. ATL was combined with epistemic logic [51, 32, 1], and several semantic variants were defined for various assumptions about agents' memory and available information [48, 32, 42]. Also, many conceptual extensions have been considered, e.g., with explicit reasoning about strategies [49, 53, 19, 41], agents with bounded resources [3, 16], and reasoning about persistent strategies and commitment [2, 14]. Especially the last kind of semantics will prove useful in our analysis.

2. PRELIMINARIES

2.1 Models and Modules

In this paper, we consider several frameworks for modeling and verification of temporal properties. *Modules* in *module checking* [35] were proposed to represent open systems – that is, systems that interact with an environment whose behavior cannot be determined in advance. In their simplest form, modules are unlabeled transition systems with the set of states partitioned into those "owned" by the system, and the ones where the next transition is controlled by the environment. Models of *alternating-time temporal logic* [8], called *concurrent game structures*, are multi-player transition systems with transitions labeled by tuples of actions, one from each agent.

[1]One can see CTL/CTL* module checking as a logical system where the syntax is simply CTL/CTL* but the semantics is altered. Thus, by a slight abuse of terminology, we can talk about the "expressive power" of module checking. We clarify the issue in Section 3.1.

DEFINITION 1 (MODULE). *A* module *is a tuple* $M = \langle AP, St_s, St_e, q_0, \rightarrow, PV\rangle$*, where* AP *is a finite set of (atomic) propositions,* $St = St_s \cup St_e$ *is a nonempty finite set of states partitioned into a set* St_s *of* system *states and a set* St_e *of* environment *states,* $\rightarrow \subseteq St \times St$ *is a (global) transition relation,* $q_0 \in St$ *is an initial state, and* $PV : St \rightarrow 2^{AP}$ *maps each state* q *to the set of atomic propositions true in* q*.*

DEFINITION 2 (CGS). *A* concurrent game structure *is a tuple* $M = \langle AP, \mathbb{A}\mathrm{gt}, St, Act, d, o, PV\rangle$ *including nonempty finite set of propositions* AP*, agents* $\mathbb{A}\mathrm{gt} = \{1, \ldots, k\}$*, states* St*, (atomic) actions* Act*, and a propositional valuation* $PV : St \rightarrow 2^{AP}$*. The function* $d : \mathbb{A}\mathrm{gt} \times St \rightarrow 2^{Act}$ *defines nonempty sets of actions available to agents at each state, and the (deterministic) transition function* o *assigns the outcome state* $q' = o(q, \alpha_1, \ldots, \alpha_k)$ *to each state* q *and tuple of actions* $\alpha_i \in d(i, q)$ *that can be executed by* $\mathbb{A}\mathrm{gt}$ *in* q*.*

A pointed CGS *is a pair* (M, q_0) *of a concurrent game structure and an initial state in it.*

Nondeterministic choices of agents in a CGS can be represented by sets of actions. In this sense, agent a can select at state q any nonempty set $\boldsymbol{\alpha} \subseteq d(a, q)$, and the set of successors of $\boldsymbol{\alpha}$ is simply the union of successor sets for each action in α. Then, modules can be seen as a subclass of concurrent game structures – more precisely, 2-player turn-based[2] pointed CGS's with agents $\mathbb{A}\mathrm{gt} = \{sys, env\}$. We will use both definitions of modules interchangeably.

To give an example, consider an ATM that allows customers to *choose* among the operations of *putting* money, *getting* money, or *checking* the balance, cf. Figure 1. The environment represents all possible infinite lines of customers, each with their own plans. Also, the machine can loop in the starting state, preventing customers from executing any operation. We formally define the ATM as a module $M_{ATM} = \langle AP, St_s, St_e, \rightarrow, q_0, PV\rangle$ such that $St_s = \{start, put, get, check\}$, $St_e = \{choice\}$, $AP = St$, $PV(x) = \{x\}$ for each $x \in AP$, $q_0 = start$ and $\rightarrow = \{(start, start), (start, choice), (choice, x), (x, start) \mid x \in St_s \backslash \{start\}\}$. We leave rewriting of M_{ATM} as a CGS for the interested reader.

2.2 CTL/CTL* Module Checking

We first recall the definition of labeled trees, and the syntax as well as semantics of *Computation Tree Logic*.

Let $\mathbb{N}$ be the set of positive integers. A *tree* T is a prefix closed subset of $\mathbb{N}^*$. The elements of T are called *nodes* and the empty word ε is the *root* of T. For $x \in T$, the set of *children* of x is $children(T, x) = \{x \cdot i \in T \mid i \in \mathbb{N}\}$. For $k \geq 1$, the (complete) k-ary tree is the tree $\{1, \ldots, k\}^*$. For $x, y \in \mathbb{N}^*$, we write $x \prec y$ to mean that x is a proper prefix of y. For $x \in T$, a (full) path λ of T from x is a *minimal* set $\lambda \subseteq T$ such that $x \in \lambda$ and for each $y \in \lambda$ such that $children(T, y) \neq \emptyset$, there is exactly one node in $children(T, y)$ belonging to λ. Given a path $\lambda = x_0 x_1 \ldots$ we denote $\lambda[0] = x_0$ and $\lambda[1..\infty] = x_1 x_2 \ldots$. For $y \in \lambda$, we denote by λ^y the (suffix) path of T from y given by $\{z \in \lambda \mid y \preceq z\}$. For an alphabet Σ, a Σ-labeled tree is a pair $\langle T, V\rangle$ where T is a tree and $V : T \rightarrow \Sigma$ maps each node of T to a symbol in Σ.

[2]A CGS is turn-based iff every state in it is controlled by (at most) one agent. That is, for every $q \in St$, there is an agent $a \in \mathbb{A}\mathrm{gt}$ such that $|d(a', q)| = 1$ for all $a' \neq a$. The agent a is the "owner" of state q. Note that, in a turn-based CGS, if we label states with their owners then there is no need to label transitions with tuples of actions anymore.

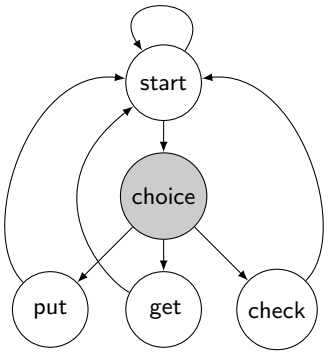

Figure 1: ATM model M_{ATM}. Environment states are marked gray; system states are marked white.

CTL* is a branching–time temporal logic [24], where path quantifiers, E ("for some path") and A ("for all paths"), can be followed by an arbitrary linear-time formula, allowing boolean combinations and nesting over temporal operators X ("next"), U ("strong until"), F ("eventually"), and G ("always"). There are two types of formulas in CTL*: *state formulas* φ, whose satisfaction is related to a specific state (or node of a labeled tree), and *path formulas* γ, whose satisfaction is related to a specific path. Formally:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{E}\gamma,$$
$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \mathsf{X}\gamma \mid \gamma\,\mathsf{U}\,\gamma.$$

where p is an atomic proposition. The other operators can be defined as: $\mathsf{A}\gamma \equiv \neg\mathsf{E}\neg\gamma$, $\mathsf{F}\gamma \equiv true\,\mathsf{U}\,\gamma$, and $\mathsf{G}\gamma \equiv \neg\mathsf{F}\neg\gamma$.

CTL [21] is a restricted subset of CTL*, obtained replacing the syntax of path formulas γ as follows: $\gamma := \mathsf{X}\varphi \mid \varphi\,\mathsf{U}\,\varphi \mid \varphi\,\mathsf{W}\,\varphi$ ("weak until"), i.e., every path quantifier must be immediately followed by a temporal operator.

We define the semantics of CTL* (and its fragment CTL) with respect to a St-labeled tree $\langle T, V\rangle$ with propositional valuation PV. Let $x \in T$ and $\lambda \subseteq T$ be a path from x. For a state (resp., path) formula φ (resp. γ), the satisfaction relation $\langle T, V\rangle, PV, x \models \varphi$ (resp., $\langle T, V\rangle, PV, \lambda \models \gamma$) is defined as follows:

- $\langle T, V\rangle, PV, x \models \mathsf{p}$ *iff* $\mathsf{p} \in PV(x)$;
- $\langle T, V\rangle, PV, x \models \mathsf{E}\gamma$ *iff* there exists a path λ from x such that $\langle T, V\rangle, PV, \lambda \models \gamma$;
- $\langle T, V\rangle, PV, \lambda \models \varphi$ *iff* $\langle T, V\rangle, PV, \lambda[0] \models \varphi$;
- $\langle T, V\rangle, PV, \lambda \models \mathsf{X}\gamma$ *iff* $\langle T, V\rangle, PV, \lambda[1..\infty] \models \gamma$;
- $\langle T, V\rangle, PV, \lambda \models \gamma_1\,\mathsf{U}\,\gamma_2$ *iff* there is $y \in \lambda$ such that $\langle T, V\rangle, PV, \lambda^y) \models \gamma_2$ and $\langle T, V\rangle, PV, \lambda^z) \models \gamma_1$ for all $z \in \lambda$ such that $z \prec y$.

The clauses for negation and conjunction are standard. In addition, $\langle T, V\rangle, PV, \lambda \models \gamma_1\,\mathsf{W}\,\gamma_2$ iff either $\langle T, V\rangle, PV, \lambda \models \gamma_1\,\mathsf{U}\,\gamma_2$ or $\langle T, V\rangle, PV, \lambda \models \mathsf{G}\gamma_1$. Given a CTL* (state) formula φ, we say that $\langle T, V\rangle$ satisfies φ if $\langle T, V\rangle, PV, \varepsilon \models \varphi$.

For a module $M = \langle AP, St_s, St_e, \rightarrow, q_0, PV\rangle$, the set of all (maximal) computations of M starting from the initial state q_0 is described by a St-labeled tree $\langle T_M, V_M\rangle$, called *computation tree*, which is obtained by unwinding M from

the initial state in the usual way. The problem of deciding, for a given branching-time formula ψ over AP, whether $\langle T_M, PV \circ V_M\rangle$ [3] satisfies ψ, denoted $M \models \psi$, is the usual *model-checking problem* [21, 45]. On the other hand, for an open system, $\langle T_M, V_M\rangle$ corresponds to a very specific environment, i.e. the maximal environment that never restricts the set of its next states. When we examine specification ψ w.r.t. a module M, the formula ψ should hold not only in $\langle T_M, V_M\rangle$, but in all the trees obtained by pruning some environment transitions from $\langle T_M, V_M\rangle$. The set of these trees is denoted by $exec(M)$ and is formally defined as follows. For each state $q \in St$, let $succ(q)$ be the ordered tuple of q' node successors of q, i.e., $q \to q'$. A tree $\langle T, V\rangle$ is in $exec(M)$ iff $T \subseteq T_M$, V is the restriction of V_M to the tree T, and for all $x \in T$ the following holds:

- if $V_M(x) = w \in St_s$ and $succ(q) = \langle q_1, \dots, q_n\rangle$, then $children(T, x) = \{x \cdot 1, \dots, x \cdot n\}$ (note that for $1 \le i \le n$, $V(x \cdot i) = V_M(x \cdot i) = q_i$);
- if $V_M(x) = w \in St_e$ and $succ(q) = \langle q_1, \dots, q_n\rangle$, then there is a sub-tuple $\langle q_{i_1}, \dots, q_{i_p}\rangle$ of $succ(q)$ such that $children(T, x) = \{x \cdot i_1, \dots, x \cdot i_p\}$ (note that for $1 \le j \le p$, $V(x \cdot i_j) = V_M(x \cdot i_j) = q_{i_j}$).

Intuitively, when the module M is in a system state q_s, then all states in $succ(q_s)$ are possible successors. When M is in an environment state q_e, then the possible next states (that are in $succ(q_e)$) depend on the current environment. Since the behavior of the environment is nondeterministic, we have to consider all the nonempty sub-tuples of $succ(q_e)$.

For a module M and a CTL(*) formula ψ, we say that M reactively satisfies ψ, denoted by $M \models^r \psi$, if all the trees in $exec(M)$ satisfy ψ. The problem of deciding whether M reactively satisfies ψ is called *module checking* [37]. Note that $M \models^r \psi$ implies $M \models \psi$ (since $\langle T_M, V_M\rangle \in exec(M)$), but the converse in general does not hold. Also, note that $M \not\models^r \psi$ is *not* equivalent to $M \models^r \neg\psi$.

As an example, consider again the ATM module from Figure 1. Clearly, $M_{ATM} \models \mathsf{EFput}$ as it is (in principle) possible to deposit money. On the other hand, $M_{ATM} \not\models^r \mathsf{EFput}$. Think of a line of people who never want to deposit. It corresponds to an execution tree of M_{ATM} with no node labeled with put, and such a tree does not satisfy EFput.

2.3 Alternating Time Logic ATL/ATL*

Alternating-time temporal logic [8] generalizes CTL* by replacing path quantifiers E, A with *strategic modalities* $\langle\langle A\rangle\rangle$. Informally, $\langle\langle A\rangle\rangle\gamma$ expresses that the group of agents A has a collective strategy to enforce temporal property γ. The language ATL* is given by the grammar below:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\langle A\rangle\rangle\gamma,$$
$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \wedge \gamma \mid \mathsf{X}\gamma \mid \gamma\,\mathsf{U}\,\gamma.$$

where $A \subseteq \mathbb{A}\mathrm{gt}$ is any subset of agents, and p is a proposition. "Sometime" and "always" are obtained like in CTL*. Also, we can use $[\![A]\!]\gamma \equiv \neg\langle\langle A\rangle\rangle\neg\gamma$ to express that no strategy of A can prevent property γ. Similarly to CTL, ATL is the syntactic variant in which every occurrence of a strategic modality is immediately followed by a temporal operator.

Given a CGS, we define the strategies and their outcomes as follows. A *perfect recall strategy* for agent a is a function $s_a : St^+ \to Act$ such that $s_a(q_0 q_1 \dots q_n) \in d(a, q_n)$. A *memoryless strategy* for a is a function $s_a : St \to Act$ such that $s_a(q) \in d(a, q)$. A *collective strategy* for a group of agents $A = \{a_1, \dots, a_r\}$ is simply a tuple of individual strategies $s_A = \langle s_{a_1}, \dots, s_{a_r}\rangle$. The "outcome" function $out(q, s_A)$ returns the set of all paths that can occur when agents A execute strategy s_A from state q on. The semantics $\models_{\mathrm{ATL}}$ of alternating-time logic is obtained from that of CTL* by replacing the clause for $\mathsf{E}\phi$ as follows:

$M, q \models \langle\langle A\rangle\rangle\gamma$ iff there is a perfect recall strategy s_A for A such that for every $\lambda \in out(q, s_A)$ we have $M, \lambda \models \gamma$.

We will refer to the logical system $(\mathrm{ATL}^*, \models_{\mathrm{ATL}})$ simply as ATL*, and to $(\mathrm{ATL}, \models_{\mathrm{ATL}})$ as ATL.The problem of deciding whether a pointed CGS (M, q_0) satisfies the ATL(*) formula ψ is called ATL(*) *model checking* problem [8].

Consider the ATM module from Figure 1 again. Clearly, $M_{ATM} \models_{\mathrm{ATL}} \langle\langle sys, env\rangle\rangle\mathsf{Fput}$. The right strategy is: for the agent *sys*, proceed from start to choice, and for the agent *env*, proceed from choice to put. On the other hand, $M_{ATM} \models_{\mathrm{ATL}} \neg\langle\langle sys\rangle\rangle\mathsf{Fput} \wedge \neg\langle\langle env\rangle\rangle\mathsf{Fput}$: none of the agents can bring about put without cooperation from the other one.

Embedding CTL* in ATL*. The path quantifiers of CTL* can be expressed in the standard semantics of ATL* as follows: $\mathsf{A}\phi \equiv \langle\langle\emptyset\rangle\rangle\mathsf{A}$ and $\mathsf{E}\phi \equiv \langle\langle\mathbb{A}\mathrm{gt}\rangle\rangle\phi$ [8]. We point out that the above translation of E does *not* work for several extensions of ATL*, e.g., with imperfect information (cf. [18]), nondeterministic strategies, and irrevocable strategies. On the other hand, the translation of A into $\langle\langle\emptyset\rangle\rangle$ does work for all the semantic variants of ATL* considered in this paper. Thanks to that, we can define a translation $atl(\phi)$ from CTL* to ATL* as follows. First, we convert ϕ so that it only includes universal path quantifiers, and then replace every occurrence of A with $\langle\langle\emptyset\rangle\rangle$. For example, $atl(\mathsf{EG}(\mathsf{p_1} \wedge \mathsf{AF p_2})) = \neg\langle\langle\emptyset\rangle\rangle\mathsf{F}(\neg\mathsf{p_1} \vee \neg\langle\langle\emptyset\rangle\rangle\mathsf{F p_2})$. Note that if ϕ is a CTL formula then $atl(\phi)$ is a formula of ATL.

3. MODULE CHECKING VS. ATL

What is the relationship between CTL *module* checking and ATL *model* checking? Intuitively, it seems that the former is a special case of the latter. $M, q \models^r \phi$ expresses that, for every possible strategy of the environment, the system can always respond in such a way that formula ϕ will hold on the resulting path(s). This suggests $[\![env]\!]atl(\phi)$ as a suitable translation of module checking into ATL. There are, however, two features that differ from the standard semantics of ATL. First, strategies of the environment are *irrevocable* in the sense of [2]. Technically, this means that the execution tree is pruned of all the transitions inconsistent with the strategy before we start to evaluate ϕ. In contrast, nested subformulae in ATL are evaluated in the original transition system. Secondly, the environment's strategies in module checking are *nondeterministic*, whereas the semantics of ATL admits only deterministic strategies. We show that these features are essential and make embedding of module checking into ATL impossible.

3.1 No Pruning, No Module Checking

Before we proceed, we briefly introduce the notions of distinguishing power and expressive power (cf. e.g. [54]).

DEFINITION 3 (DISTINGUISHING AND EXPRESSIVE POWER). *Let $L_1 = (\mathcal{L}_1, \models_1)$ and $L_2 = (\mathcal{L}_2, \models_2)$ be two logical systems*

[3] $PV \circ V_M$ denotes the composition of the functions PV and V_M that allows to re-label each state-labeled node u in $\langle T_M, V_M\rangle$ with $PV(V_M(u))$.

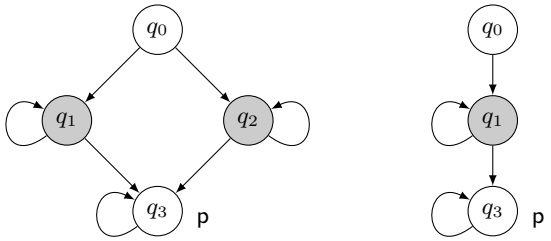

Figure 2: Distinguishing power of pruning: modules M_1 (left) and M_2 (right)

over the same class of models $\mathcal{M}$. By $[\![\phi]\!]_{\models} = \{(M,q) \mid M,q \models \phi\}$, we denote the class of pointed models that satisfy ϕ in the semantics given by $\models$. Likewise, $[\![\phi, M]\!]_{\models} = \{q \mid M,q \models \phi\}$ is the set of states (or, equivalently, pointed models) that satisfy ϕ in a given structure M.

L_2 is at least as expressive as L_1 (written: $L_1 \preceq_e L_2$ iff for every formula $\phi_1 \in \mathcal{L}_1$ there exists $\phi_2 \in \mathcal{L}_2$ such that $[\![\phi_1]\!]_{\models_1} = [\![\phi_2]\!]_{\models_2}$.

L_2 is at least as distinguishing as L_1 (written: $L_1 \preceq_d L_2$ iff for every model M and formula $\phi_1 \in \mathcal{L}_1$ there exists $\phi_2 \in \mathcal{L}_2$ such that $[\![\phi_1, M]\!]_{\models_1} = [\![\phi_2, M]\!]_{\models_2}$.[4]

Note that $L_1 \preceq_e L_2$ implies $L_1 \preceq_d L_2$ but the converse is not true. For example, it is known that CTL has the same distinguishing power as CTL*, but strictly less expressive power. We also observe that module checking CTL* can be seen as a logical system (CTL*, $\models^r$), and analogously for module checking CTL. Thus, we can use Definition 3 to study the expressivity of both problems.

THEOREM 1. *ATL* is not sufficient to express CTL/CTL* module checking. Formally, $(CTL, \models^r) \not\preceq_d ATL^*$, and hence also $(CTL, \models^r) \not\preceq_e ATL^*$.*

Proof. We adapt an example from [2], see Figure 2. It is easy to check that (M_1, q_0) and (M_2, q_0) are in strategic bisimulation [2]. Thus, they must satisfy exactly the same formulae of ATL*. On the other hand, for the CTL formula $\phi \equiv \mathsf{AXEXp} \vee \mathsf{AXEX}\neg\mathsf{p}$, we have that $M_1, q_0 \not\models^r \phi$ but $M_2, q_0 \models^r \phi$, which concludes the proof. ■

Note that the proof does not use the fact that ATL admits only deterministic strategies. Clearly, extending the semantics of ATL to nondeterministic strategies would not change the above result. The crucial factor is pruning of the model according to the strategy of the environment. In particular, it is easy to see why the "intuitive" translation of $M,q \models^r \phi$ to $M,q \models_{\text{ATL}} [\![env]\!]atl(\phi)$ does not work without pruning. Consider again $\phi \equiv \mathsf{AXEXp} \vee \mathsf{AXEX}\neg\mathsf{p}$. In ATL, selected strategies are "forgotten" when a nested strategic modality is encountered. Hence, the formula $\langle\!\langle a \rangle\!\rangle(\langle\!\langle b \rangle\!\rangle \psi_1 \vee \langle\!\langle c \rangle\!\rangle \psi_2) \leftrightarrow (\langle\!\langle b \rangle\!\rangle \psi_1 \vee \langle\!\langle c \rangle\!\rangle \psi_2)$ is valid in ATL*. In consequence, we have $M,q \models_{\text{ATL}} [\![env]\!]atl(\phi)$ iff $M,q \models_{\text{ATL}} atl(\phi)$ iff $M,q \models_{\text{CTL}} \phi$, which is certainly *not* equivalent to $M,q \models^r \phi$.

3.2 Pruning and Irrevocable Strategies

[4]Equivalently: for every pair of pointed models that can be distinguished by some $\phi_1 \in \mathcal{L}_1$ there exists $\phi_2 \in \mathcal{L}_2$ that distinguishes these models.

Strategies in ATL* are *revocable* in the sense that in the evaluation of a nested strategic modality, agents are no longer restricted by strategies they have possibly chosen in order to reach the state where the nested modality is evaluated. As an example, consider formula $\langle\!\langle os \rangle\!\rangle \mathsf{G}\langle\!\langle a \rangle\!\rangle \mathsf{Fprint}$ that says that the operating system is able to provide the printing facility to process a. In ATL*, the formula is harder to satisfy than one might expect. In particular, when evaluating $\phi \equiv \langle\!\langle a \rangle\!\rangle \mathsf{Fprint}$ at an intermediate state, we must find a strategy for a that works *against* any possible behavior of os – despite the fact that os has selected its strategy in $\langle\!\langle os \rangle\!\rangle \mathsf{G}\phi$ exactly to make ϕ true.

A different semantics of strategic play was considered in [2]. There, strategies are assumed *irrevocable* in that they *completely* specify the player's behaviour in all conceivable situations, and for all future moments. Semantically, this can be implemented by a model update that prunes from the model all the transitions that cannot occur if players A execute strategy s_A.

DEFINITION 4 (MODEL UPDATE). *Let M be a CGS, A a coalition, and s_A a memoryless strategy of A. The update of M by s_A, denoted $M \dagger s_A$, is the same as M, except that the choices of each agent $a \in A$ are fixed by the strategy, i.e., $d(a,q) = \{s_a(q)\}$ for each $a \in A$ and $q \in St$.*

The semantics $\models_{\text{IATL}}$ of "Irrevocable ATL" replaces the clause for strategic modalities as follows:

$M,q \models_{\text{IATL}} \langle\!\langle A \rangle\!\rangle \phi$ iff there is a memoryless strategy s_A for A st. for every $\lambda \in out(q, s_A)$ we have $M \dagger s_A, \lambda \models \phi$.

Moreover, the "irrevocable" semantics for agents with perfect memory extends $\models_{\text{IATL}}$ with the clause:

$M,q \models_{\text{MIATL}} \phi$ iff $tree(M,q), q \models_{\text{IATL}} \phi$,

where $tree(M,q)$ is the tree unfolding of the pointed CGS (M,q). We will refer to the logical system $(\mathcal{L}_{ATL^*}, \models_{\text{MIATL}})$ as MIATL*, and to $(\mathcal{L}_{ATL}, \models_{\text{MIATL}})$ as MIATL. We note in passing that MIATL can be seen as a special case of two more expressive languages: ATL with strategy contexts [14] and Strategy Logic [41].

The difference between revocable and irrevocable strategies shows best when we consider the ability to achieve a goal which, again, involves (nested) strategic ability. For example, formula $\langle\!\langle a \rangle\!\rangle \mathsf{G}(\mathsf{p} \wedge \langle\!\langle a \rangle\!\rangle \mathsf{X}\neg\mathsf{p})$ is satisfiable in the standard semantics of ATL, but unsatisfiable in IATL and MIATL.

As it turns out, changing the semantics of ATL to irrevocable strategies gives to the logic the whole *distinguishing* power of module checking. On the other hand, we strongly suspect that MIATL is still not enough to cover the whole *expressive* power of module checking.

THEOREM 2. *MIATL has at least the distinguishing power of CTL* module checking.*

Proof. We prove that if MIATL does not distinguish two models then module checking also does not. If two pointed models M_1, M_2 satisfy the same formulae of MIATL then they must be in alternating bisimulation [6], and hence also their tree unfoldings are alternating-bisimilar. But then, for every $T \in exec(M_1)$ there must be a bisimilar $T' \in exec(M_2)$, and vice versa. Thus, T and T' must satisfy exactly the same formulae of CTL*. In consequence, $M_1 \models^r \phi$ iff $\forall T \in exec(M_1).T \models_{\text{CTL}^*} \phi$ iff $\forall T' \in exec(M_2).T_2 \models_{\text{CTL}^*} \phi$ iff $M_2 \models^r \phi$. ■

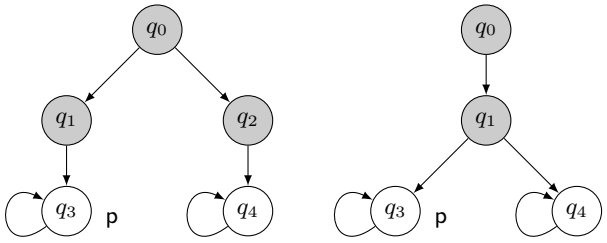

Figure 3: Distinguishing power of nondeterministic strategies: modules M_3 (left) and M_4 (right)

CONJECTURE 1. *MIATL does not cover the expressive power of CTL*/CTL module checking.*

Proof sketch. Let $\phi \equiv \phi_1 \to \phi_2$, where $\phi_1 \equiv \mathsf{AGAFp}$, and $\phi_2 \equiv \mathsf{AG}(\mathsf{EXp} \wedge \mathsf{EX}\neg\mathsf{p})$. We argue that there is no formula ϕ' of MIATL* such that $M, q \models^r \phi$ iff $M, q \models_{\text{MIATL}} \phi'$.

First, ϕ_1 is equivalent to the CTL* formula AGFp expressing that p occurs infinitely often on all paths (of the given tree). Secondly, ϕ_2 holds only if: (a) at every state of the model, reachable in the given tree, there is a transition to a state satisfying p and another one to a state with $\neg$p; and (b) the tree is obtained from a globally nondeterministic strategy of the environment, in the sense that at no reachable state the environment chooses deterministically. This is because, if *env* used a deterministic strategy, then at all states controlled by *env* there would be only one transition – either to a state with p, or to one with $\neg$p. Thus $M, q \models^r \phi$ expresses that every strategy of the environment that provides p infinitely often must be globally nondeterministic. It seems that this property cannot be expressed in MIATL, which can only quantify over *deterministic* strategies. ■

Regardless of the general (in)expressivity result, we will now show that module checking cannot be embedded in MIATL *in a straightforward way*, unless nondeterministic strategies are added to the semantics.

3.3 Deterministic Strategies Are Not Enough

We will begin by defining formally what we take as "straightforward" translations of module checking.

DEFINITION 5 (UPSAMIATL*). *We define the* universally prefixed single-agent MIATL* *(*upsaMIATL**) as the following syntactic restriction on MIATL*: only* 1 *agent name can occur in the formula, and only once, in a universal strategic modality at the beginning of the formula. That is, formulae of upsaMIATL* take form $[\![a]\!]\psi$ where a is an agent and ψ contains no strategic modalities except for $\langle\!\langle\emptyset\rangle\!\rangle$.*

Universally prefixed single-agent MIATL *(*upsaMIATL *for short) is defined as the analogous restriction of MIATL.*

We show now that upsaMIATL* is not sufficient to characterize CTL/CTL* module checking.

THEOREM 3. *upsaMIATL* covers neither the distinguishing nor expressive power of CTL/CTL* module checking. Formally, $(CTL, \models^r) \not\preceq_d$ upsaMIATL*, and hence also $(CTL, \models^r) \not\preceq_e$ upsaMIATL*.*

Proof. Consider modules M_3, M_4 in Figure 3. In each of them, the environment has two deterministic strategies (go "left" or go "right"). Note also that the "left" strategies yield bisimilar execution trees in M_3 and M_4, and hence the trees satisfy exactly the same formulae of CTL* (likewise for the "right" strategies). Thus, for every MIATL* formula of shape $\phi \equiv [\![env]\!]\psi$ with ψ containing no strategic modalities except $\langle\!\langle\emptyset\rangle\!\rangle$, either ϕ holds in both M_3, q_0 and M_4, q_0, or in none of them.

On the other hand, consider $\phi \equiv \mathsf{AX}(\mathsf{AXp} \vee \mathsf{AX}\neg\mathsf{p})$. It is easy to see that $M_3, q_0 \models^r \phi$ but $M_4, q_0 \not\models^r \phi$. ■

The proof shows clearly that upsaMIATL* lacks the ability to refer to nondeterministic strategies, which are a natural element of module checking. In the final step of our analysis, we will extend the semantics of MIATL* with nondeterministic strategies, and show that the result corresponds to the module checking problem.

DEFINITION 6 (MNIATL*). Nondeterministic MIATL* *(*MNIATL* for short) is given by the syntax of ATL* and the semantic relation $\models_{MNIATL}$, defined as follows. First, a nondeterministic memoryless strategy of agent a is a function $s_a : St^+ \to 2^{Act}$ such that $\emptyset \neq s_a(q_0 q_1 \dots q_k) \subseteq d(a, q_k)$. Then, we define the semantics of "nondeterministic irrevocable ATL" by extending $\models_{IATL}$ with the following clause:*

$$M, q \models_{NIATL} \langle\!\langle A\rangle\!\rangle\phi \quad \text{iff there is a memoryless strategy } s_A \text{ st. for every } \lambda \in out(q, s_A) \text{ we have } M \dagger s_A, \lambda \models \phi.$$

Finally, we define $M, q \models_{MNIATL} \phi$ iff $tree(M, q), q \models_{MIATL} \phi$, where $tree(M, q)$ is the tree unfolding of the pointed CGS (M, q). MNIATL, upsaMNIATL, and upsaMNIATL are defined as suitable syntactic restrictions of MNIATL*.*

THEOREM 4. *Module checking CTL* corresponds precisely to the universally prefixed single-agent MNIATL*. Module checking CTL corresponds precisely to the universally prefixed single-agent MNIATL.*

Proof. **(i) module checking $\preceq_e$ MNIATL.** We use the following translation for module checking of CTL*: $tr(\phi) = [\![env]\!]atl(\phi)$. Now, $M, q \models^r \phi$ iff for every $T \in exec(M, q)$ we have $T \models_{\text{CTL}} \phi$. Since every such tree corresponds to a nondeterministic memoryless strategy in $tree(M, q)$, this is equivalent to saying that $\big(tree(M, q) \dagger s_{env}\big), q \models_{\text{NIATL}} atl(\phi)$ for every such strategy s_{env}. Thus, equivalently, we have that $M, q \models_{\text{MNIATL}} [\![env]\!]atl(\phi)$.

Note that $[\![env]\!]atl(\phi)$ is not a formula of MNIATL (without star) even if ϕ is a formula of CTL. For example, for $\phi \equiv \mathsf{AFp}$, we get $[\![env]\!]\langle\!\langle\emptyset\rangle\!\rangle\mathsf{Fp}$. However, if ϕ is in CTL, we can improve the translation as follows: $tr'(\phi) = [\![env]\!]\mathsf{Now}\, atl(\phi)$ where $\mathsf{Now}\,\phi \equiv \phi\,\mathsf{U}\,\phi$. For example, $tr'(\mathsf{AFp})$ becomes $[\![env]\!](\langle\!\langle\emptyset\rangle\!\rangle\mathsf{Fp})\,\mathsf{U}\,(\langle\!\langle\emptyset\rangle\!\rangle\mathsf{Fp})$, which *is* a MNIATL formula.

Clearly, given a state formula ϕ and a path λ, we have $\lambda \models \mathsf{Now}\,\phi$ iff $\lambda[0] \models \phi$ in any semantics $\models$ used in this paper. Thus, $M, q \models_{\text{MNIATL}} [\![env]\!]\mathsf{Now}\, atl(\phi)$ iff $M, q \models_{\text{MNIATL}} [\![env]\!]\mathsf{Now}\, atl(\phi)$, which proves that tr' is a correct translation of CTL module checking into MNIATL.

We also observe that the outcome of tr, as well as tr', is always in the universally prefixed single-agent fragment of the language.

(ii) MNIATL $\preceq_e$ module checking. We take a formula of universal single-agent prefixed MNIATL* $\phi \equiv [\![a]\!]\psi$, and we obtain ψ' by replacing all occurrences of $\langle\!\langle\emptyset\rangle\!\rangle$ with A. We also fix a to play the role of the environment. Then, $M, q \models_{\text{MNIATL*}} \phi$ iff $M, q \models^r \mathsf{A}\psi'$ (and likewise for "vanilla" MNIATL). ■

3.4 Discussion

Our results show in a formal way that – and how – the original version of ATL* is *not* the right framework to embed module checking:

1. Most importantly, Theorem 1 shows that there are properties expressible via module checking, which cannot be expressed in ATL*. This is the case even if the model is given (i.e., in the sense of distinguishing power). The proof also suggests that the reason lies in lack of pruning of the non-selected choices from the model (on the side of ATL*).
2. Indeed, changing the semantics of ATL* to one based on pruning (MIATL*) is a step in the right direction, as the MIATL* covers the whole distinguishing power of CTL* module checking, cf. Theorem 2. This is also confirmed by known complexity results for MIATL [14] that match with those of CTL module checking [35].
3. Still, it seems that pruning is not enough to have a general, model-independent translation of module checking into alternating-time logic (Conjecture 1). We also prove that CTL/CTL* module checking does not admit a *natural* translation into MIATL* even in the sense of distinguishing power, i.e., when a model is given (Theorem 3).
4. Finally, adding nondeterministic strategies on top of pruning allows to embed the whole CTL* module checking into the framework of alternating-time logic.

Thus, we conclude that both irrevocability and nondeterminism of environment's strategies are *essential* features of module checking.

4. CONCLUSIONS

In this paper, we have formally addressed the relationship between CTL*/CTL module checking and ATL*/ATL model checking. Since their early introduction, dating back to 1996/1997, there has been a common belief that the latter would subsume the former in a straightforward way. In this paper we show that this cannot be done. The main reason lies in the fact that in module checking strategies of the environment are nondeterministic and irrevocable (formally represented by pruning a part of the computation tree). In ATL*, instead, agents can only use deterministic and revocable strategies. We prove that – due to these limitations – ATL* model checking does not cover the distinguishing and expressive power of CTL* module checking, and even module checking of the less expressive logic CTL. We show that the lack of distinguishing power crucially stems from revocability of strategies in ATL*. Indeed, by considering the MIATL* extension of ATL* in which strategies are irrevocable, we show that a variant of ATL* model checking with at least the same distinguishing power as CTL* module checking. On the other hand, we show that module checking cannot be embedded in MIATL* in a natural way, because the latter lacks nondeterministic strategies. Finally, we present a syntactic and semantic variant of ATL* that exactly corresponds to the problem of module checking CTL/CTL* specifications.

In the recent years, a large effort has been devoted to introduce and investigate extensions of ATL* in order to come up with logics that properly address specific formal verification scenarios. Individuating the right logic and its peculiarities in correspondence of a precise decision problem is always a challenging task. This paper shows some advantages of that research. We have been able to grasp the right ingredients one has to add to ATL* in order to properly simulate CTL* module checking, thanks to the existing literature on revocability of strategies and strategic commitment.

Finally, our results open some questions for future work in practical aspects of verification of open systems. Indeed, the belief that ATL* model checking was a straightforward multi-agent extension of CTL* module checking has somehow reduced the development of module checking tools. Research concentrated mainly on ATL* model checkers, assuming that those can be used also for module checking. As we have now shown, that is not possible; in consequence, there is a gap to fill. This is extremely important since module checking has the right power to address reliability of a system embedded in hostile, or simply unpredictable environment. Another interesting question concerns comparison of module and model checking for systems with incomplete information. Both frameworks have their semantic variants based on the assumption of partial observability. We plan to study their relationship in the future.

Acknowledgements. Wojciech Jamroga acknowledges the support of the FNR (National Research Fund) Luxembourg under project GALOT – INTER/DFG/12/06.

Aniello Murano acknowledges the support of the FP7 EU project 600958-SHERPA, the IndAM project "Logiche di Gioco Estese", and the OR.C.HE.S.T.R.A. PON project.

5. REFERENCES

[1] T. Ågotnes. Action and knowledge in alternating-time temporal logic. *Synthese*, 149(2):377–409, 2006. Section on Knowledge, Rationality and Action.

[2] T. Ågotnes, V. Goranko, and W. Jamroga. Alternating-time temporal logics with irrevocable strategies. In *Proceedings of TARK XI*, pages 15–24. Presses Universitaires de Louvain, 2007.

[3] N. Alechina, B. Logan, H. Nguyen, and A. Rakib. Resource-bounded alternating-time temporal logic. In *Proceedings of AAMAS*, pages 481–488, 2010.

[4] R. Alur, L. de Alfaro, R. Grossu, T. Henzinger, M. Kang, C. Kirsch, R. Majumdar, F. Mang, and B.-Y. Wang. jMocha: A model-checking tool that exploits design structure. In *Proceedings of ICSE*, pages 835–836. IEEE Computer Society Press, 2001.

[5] R. Alur, L. de Alfaro, T. A. Henzinger, S. Krishnan, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. Technical report, University of Berkeley, 2000.

[6] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *Proceedings of CONCUR*, LNCS 1466, pages 163–178. Springer, 1998.

[7] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 100–109. IEEE Computer Society Press, 1997.

[8] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.

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

[10] B. Aminof, A. Murano, and M. Vardi. Pushdown module checking with imperfect information. In *Proceedings of*

CONCUR, LNCS 4703, pages 461–476. Springer, 2007.

[11] S. Basu, P. S. Roop, and R. Sinha. Local module checking for ctl specifications. *Electronic Notes in Theoretical Computer Science*, 176(2):125–141, 2007.

[12] L. Bozzelli. New results on pushdown module checking with imperfect information. In *Proceedings of GandALF*, volume 54 of *EPTCS*, pages 162–177, 2011.

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

[14] T. Brihaye, A. D. C. Lopes, F. Laroussinie, and N. Markey. ATL with strategy contexts and bounded memory. In *Proceedings of LFCS*, LNCS 5407, pages 92–106. Springer, 2009.

[15] N. Bulling, J. Dix, and W. Jamroga. Model checking logics of strategic ability: Complexity. In *Specification and Verification of Multi-Agent Systems*, pages 125–159. Springer, 2010.

[16] N. Bulling and B. Farwer. On the (un-)decidability of model checking resource-bounded agents. In *Proceedings of ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 567–572. IOS Press, 2010.

[17] N. Bulling and W. Jamroga. Verifying agents with memory is harder than it seemed. *AI Communications*, 23:380–403, 2010.

[18] N. Bulling and W. Jamroga. Comparing variants of strategic ability. *Journal of Autonomous Agents and Multi-Agent Systems*, 2013.

[19] K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In *Proceedings of CONCUR*, LNCS 4703, pages 59–73. Springer, 2007.

[20] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In *Proceedings of TACAS*, LNCS 7795, pages 185–191. Springer, 2013.

[21] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of Logics of Programs Workshop*, LNCS 131, pages 52–71. Springer, 1981.

[22] M. Dastani and W. Jamroga. Reasoning about strategies of multi-agent programs. In *Proceedings of AAMAS2010*, pages 625–632, 2010.

[23] L. de Alfaro, P. Godefroid, and R. Jagadeesan. Three-valued abstractions of games: Uncertainty, but with precision. In *Proceedings of LICS*, pages 170–179. IEEE Computer Society, 2004.

[24] E. Emerson and J. Halpern. "sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[25] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, 1990.

[26] A. Ferrante, A. Murano, and M. Parente. Enriched μ-calculi module checking. *Logical Methods in Computer Science*, 4(3:1):1–21, 2008.

[27] M. Gesell and K. Schneider. Modular verification of synchronous programs. In *Proceedings of ACSD*, pages 70–79. IEEE, 2013.

[28] P. Godefroid. Reasoning about abstract open systems with generalized module checking. In *Proceedings of EMSOFT*, LNCS 2855, pages 223–240. Springer, 2003.

[29] P. Godefroid and M. Huth. Model checking vs. generalized model checking: Semantic minimizations for temporal logics. In *Proceedings of LICS*, pages 158–167. IEEE Computer Society, 2005.

[30] W. Jamroga and J. Dix. Do agents make model checking explode (computationally)? In *Proceedings of CEEMAS 2005*, LNCS 3690, pages 398–407. Springer, 2005.

[31] W. Jamroga, S. Mauw, and M. Melissen. Fairness in non-repudiation protocols. In *Proceedings of STM'11*, LNCS 7170, pages 122–139. Springer 2012.

[32] W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 63(2–3):185–219, 2004.

[33] M. Kacprzak and W. Penczek. Unbounded model checking for Alternating-time Temporal Logic. In *Proceedings of AAMAS-04*, 2004.

[34] S. Kremer and J.-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. *Journal of Computer Security*, 11(3), 2003.

[35] O. Kupferman and M. Vardi. Module checking. In *Proceedings of CAV*, LNCS 1102, pages 75–86. Springer, 1996.

[36] O. Kupferman and M. Vardi. Module checking revisited. In *Proceedings of CAV*, LNCS 1254, pages 36–47. Springer, 1997.

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

[38] A. Lomuscio and F. Raimondi. MCMAS : A model checker for multi-agent systems. In *Proceedings of TACAS*, LNCS 4314, pages 450–454. Springer, 2006.

[39] F. Martinelli. Module checking through partial model checking. Technical report, CNR Roma - TR-06, 2002.

[40] F. Martinelli and I. Matteucci. An approach for the specification, verification and synthesis of secure systems. *Electronic Notes in Theoretical Computer Science*, 168:29–43, 2007.

[41] F. Mogavero, A. Murano, and M. Vardi. Reasoning about strategies. In *Proceedings of FSTTCS*, pages 133–144. LIPIcs, 2010.

[42] F. Mogavero, A. Murano, and M. Vardi. Relentful Strategic Reasoning in Alternating-Time Temporal Logic. In *Proceedings of LPAR*, LNCS 6355, pages 371–386. Springer, 2010.

[43] A. Murano, M. Napoli, and M. Parente. Program complexity in hierarchical module checking. In *Proceedings of LPAR*, LNCS 5330, pages 318–332. Springer, 2008.

[44] M. Pauly. A modal logic for coalitional power in games. *Journal of Logic and Computation*, 12(1):149–166, 2002.

[45] J. Queille and J. Sifakis. Specification and verification of concurrent programs in Cesar. In *Symposium on Programming*, LNCS 137, pages 337–351. Springer, 1981.

[46] F. Raimondi. *Model Checking Multi-Agent Systems*. PhD thesis, University College London, 2006.

[47] P. Schnoebelen. The complexity of temporal model checking. In *Advances in Modal Logics, Proceedings of AiML 2002*. World Scientific, 2003.

[48] P. Y. Schobbens. Alternating-time logic with imperfect recall. *Electronic Notes in Theoretical Computer Science*, 85(2):82–93, 2004.

[49] W. van der Hoek, W. Jamroga, and M. Wooldridge. A logic for strategic reasoning. In *Proceedings of AAMAS'05*, pages 157–164, 2005.

[50] W. van der Hoek, A. Lomuscio, and M. Wooldridge. On the complexity of practical ATL model checking. In *Proceedings of AAMAS'06*, pages 201–208, 2006.

[51] W. van der Hoek and M. Wooldridge. Cooperation, knowledge and time: Alternating-time Temporal Epistemic Logic and its applications. *Studia Logica*, 75(1):125–157, 2003.

[52] D. Walther, C. Lutz, F. Wolter, and M. Wooldridge. ATL satisfiability is indeed EXPTIME-complete. *Journal of Logic and Computation*, 16(6):765–787, 2006.

[53] D. Walther, W. van der Hoek, and M. Wooldridge. Alternating-time temporal logic with explicit strategies. In *Proceedings TARK XI*, pages 269–278. Presses Universitaires de Louvain, 2007.

[54] Y. Wang and F. Dechesne. On expressive power and class invariance. *CoRR*, abs/0905.4332, 2009.