# Module Checking for Uncertain Agents

Wojciech Jamroga[1] and Aniello Murano[2]

[1] Institute of Computer Science, Polish Academy of Sciences, Poland
[2] Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione, Università degli Studi di Napoli Federico II, Italy
`w.jamroga@ipipan.waw.pl, aniello.murano@unina.it`

**Abstract.** *Module* checking is a decision problem proposed in late 1990s to formalize verification of open systems, i.e., systems that must adapt their behavior to the input they receive from the environment. It was recently shown that module checking offers a distinctly different perspective from the better-known problem of *model* checking. Module checking has been studied in several variants. Syntactically, specifications in temporal logic CTL and strategic logic ATL have been used. Semantically, the environment was assumed to have either perfect or imperfect information about the global state of the interaction. In this work, we rectify our approach to imperfect information module checking from the previous paper. Moreover, we study the variant of module checking where also the system acts under uncertainty. More precisely, we assume that the system consists of one or more agents whose decision making is constrained by their observational capabilities. We propose an automata-based verification procedure for the new problem, and establish its computational complexity.

**Keywords:** module checking, strategic logic, imperfect information

## 1 Introduction

*Module checking* [20, 22] is a formal method to automatically check for correctness of *open systems*. 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 transition system with states partitioned into ones controlled by the system and by the environment. The environment represents an external source of nondeterminism, because at each state controlled by the environment the computation can continue with any subset of its possible successor states. In consequence, we have an infinite number of computation trees to handle, one for each possible behavior of the environment. Properties for module checking are usually specified in temporal logics CTL or CTL* [9, 11].

It was believed for a long time that module checking of CTL/CTL* is a special (and rather simplistic) case of *model* checking strategic logics ATL/ATL* [2]. Because of that, active research on module checking subsided shortly after its conception. The belief has been recently refuted in [17]. There, it was proved that module checking includes two features inherently absent in the semantics of ATL, namely irrevocability and nondeterminism of strategies. This made module checking an interesting formalism for verification of open systems again.

In [18], we extended module checking to handle specifications in the more expressive logic ATL. However, [18] focused on modules of perfect information, i.e., ones where all the participants have, at any moment, complete and accurate knowledge of the current global state of the system. The assumption is clearly unrealistic, as almost all agents must act under uncertainty. In this paper, we focus on that aspect, and investigate verification of open systems that include uncertain agents. In fact, our study in [18] mentioned systems where the environment might have imperfect information. However, our treatment of such scenarios did not really capture the feasible patterns of behavior that can be produced by uncertain environments. Here, we give a new interpretation to the problem. Moreover, we generalize ATL module checking to modules that include uncertainty also on the part of the system. Finally, we investigate formal properties of the new problem in terms of expressive power, automata-based algorithms, and computational complexity.

**Related work.** Module checking was introduced in [20, 22], and later extended in several directions. In [21], 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. [13, 3] extended module checking to $\mu$-calculus specifications, and in [26] the module checking problem was investigated for bounded pushdown modules (or *hierarchical modules*). Recently, module checking was also extended to specifications in alternating-time temporal logics ATL/ATL* [18]. From a more practical point of view, [24, 25] built a semi-automated tool for module checking in the existential fragment of CTL, both in the perfect and imperfect information setting. Moreover, an approach to CTL module checking based on tableau was exploited in [5]. Finally, an extension of module checking was used to reason about three-valued abstractions in [15, 10, 16, 14].

It must be noted that literature on module checking became rather sparse after 2002. This should be partially attributed to the popular belief that CTL module checking is nothing but a special case of ATL model checking. The belief has been refuted only recently [17], which will hopefully spark renewed interest in verification of open systems by module checking.

## 2 Verification of Open Multi-Agent Systems

We first recall the main concepts behind module checking of multi-agent systems.

### 2.1 Models and Modules

*Modules* in *module checking* [20] were proposed to represent open systems – that is, systems that interact with an environment whose behavior cannot be determined in advance. Examples of modules include: an ATM interacting with customers, a steel factory depending on fluctuations in iron supplies, a Mars explorer adapting to the weather conditions, and so on. In their simplest form, modules are represented by 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.

**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 $PV : St \rightarrow 2^{AP}$ is a valuation of atomic propositions that maps each state $q$ to the set of atomic propositions that are true in $q$.*

Modules can be seen as a subclass of more general models of interaction, called *concurrent game structures* [2].

**Definition 2 (CGS).** *A* concurrent game structure (CGS) *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 $q$.*

*We will write $d_i(q)$ instead of $d(i, q)$, and denote the set of collective choice of group $A$ at state $q$ by $d_A(q) = \prod_{i\in A} d_i(q)$. We will also use $AP^M, \mathbb{A}\mathrm{gt}^M, St^M$ etc. to refer to the components of $M$ whenever confusion can arise.*

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

## 2.2 Multi-Agent Modules

Multi-agent modules have been proposed in [18] to allow for reasoning about open systems that are themselves implemented as a composition of several autonomous processes.

**Definition 3 (Multi-agent module).** *A* multi-agent module *is a pointed concurrent game structure that contains a special agent called "the environment" ($e \in \mathbb{A}\mathrm{gt}$). We call a module $k$-*agent *if it consists of $k$ agents plus the environment (i.e., the underlying CGS contains $k + 1$ agents).*

*The module is* alternating *iff its states are partitioned into those owned by the environment (i.e., $|d(a, q)| = 1$ for all $a \neq e$) and those where the environment is passive (i.e., $|d(e, q)| = 1$). That is, it alternates between the agents' and the environment's moves. Moreover, the module is* turn-based *iff the underlying CGS is turn-based.*[3]

[3] 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 $a' \neq a$.

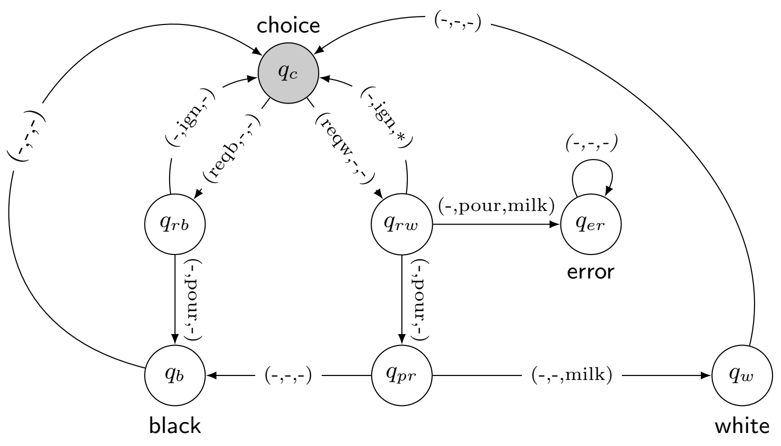

**Fig. 1.** Multi-agent coffee machine $M_{caf}$

We note in passing that the original modules from [20] were turn-based (and hence also alternating). On the other hand, the version of module checking for imperfect information in [21] assumed that the system and the environment can act simultaneously.

*Example 1.* A multi-agent coffee machine is presented in Figure 1. The module includes two agents: the brewer (*br*) and the milk provider (*milky*). The brewer's function is to pour coffee into the cup (action *pour*), and the milk provider can add milk on top (action *milk*). Moreover, each of them can be faulty and ignore the request from the environment (*ign*). Note that if *br* and *milky* try to pour coffee and milk at the same time, the machine gets to an error state. Finally, the environment has actions $reqb, reqw$ available in state $q_c$, meaning that it requests black (resp. white) coffee. Since the module is alternating, we adhere to the popular convention of marking system states as white, and environment states as grey.

## 2.3 Module Checking

The generic module checking problem can be defined as follows. Assume a modal logic $\mathcal{L}$ whose formulae are interpreted in pointed concurrent game structures according to the semantic relation $\models_{\mathcal{L}}$.[4] For example, $\mathcal{L}$ can be the computation tree logic CTL [9, 11] or alternating-time temporal logic ATL [2]. Given a CGS $M$, the set of all infinite computations of $M$ starting from the initial state $q_0$ is described by an $St$-labeled tree that we call *the computation tree of* $M$ and

[4] We will omit the subscript whenever it is clear from the context.

denote by $tree(M)$. The tree is obtained by unwinding $M$ from $q_0$ in the usual way. We omit the formal construction for lack of space, and refer the interested reader to [20, 17]. By $exec(M)$, we denote the set of all the trees obtained by pruning some environment choices from $tree(M)$ in such a way that, for each node in the tree, at least one choice remains. Note that, from a mathematical point of view, every tree $T \in exec(M)$ is an infinite pointed concurrent game structure with the same set of agents as $M$, and nodes in $St^T$ corresponding to (some) sequences of states from $St^M$. The extent of the pruning is encoded in the actual set of nodes $St^T$ and the availability function $d_e^T$ that captures the actions available to the environment in the nodes of the tree. Formally, $T_1$ is a pruning of $T_2$ iff: (i) $St^{T_1} \subseteq St^{T_2}$, (ii) $Act^{T_1} \subseteq Act^{T_2}$, (iii) for every $v \in St^{T_1}, a \neq e$, we have $d_a^{T_1}(v) = d_a^{T_2}(v)$ and $\emptyset \neq d_e^{T_1}(v) \subseteq d_e^{T_2}(v)$, (iv) $o^{T_1} = (o^{T_2} \mid St^{T_1})$, (v) $PV^{T_1} = (PV^{T_2} \mid St^{T_1})$, and (vi) the root of $T_1$ is the same as the root of $T_2$.

**Definition 4 (Module checking).** *For a pointed CGS $(M, q_0)$ and a formula $\varphi$ of logic $\mathcal{L}$, we say that $(M, q_0)$* reactively satisfies *$\varphi$, denoted by $M, q_0 \models^r_{\mathcal{L}} \varphi$, iff for every tree $T \in exec(M)$ we have that $T \models^r_{\mathcal{L}} \varphi$. Again, we will omit subscripts if they are clear from context. The problem of deciding whether $M$ reactively satisfies $\varphi$ is called* module checking *[22].*

Note that, for most modal logics, $M \models^r_{\mathcal{L}} \varphi$ implies $M \models_{\mathcal{L}} \varphi$ but the converse does not hold. Also, $M \not\models^r_{\mathcal{L}} \varphi$ is in general not equivalent to $M \models^r_{\mathcal{L}} \neg\varphi$.

*Example 2.* Consider the coffee machine from Example 1 with the CTL specification $\mathsf{EFwhite}$ saying that there exists at least one possible path where eventually white coffee will be served. Clearly, $M_{caf} \models_{CTL} \mathsf{EFwhite}$. On the other hand, $M_{caf} \not\models^r_{CTL} \mathsf{EFwhite}$. Think of a line of customers who never order white coffee. It corresponds to an execution tree of $M_{caf}$ that prunes off all nodes labeled with $q_{rw}$, and such a tree cannot satisfy $\mathsf{EFwhite}$.

## 2.4 Reasoning about Strategic Behavior: Alternating Time Logic

Alternating-time temporal logic ATL/ATL* [2] generalizes the branching-time logic CTL/CTL* [9, 11] by means of *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 $\gamma$. The language ATL* is given by the grammar below:

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

where $A \subseteq \mathbb{A}\mathrm{gt}$ is any subset of agents, and $\mathsf{p}$ is a proposition. Temporal operators X, U stand for "next" and "until", respectively. The "sometime" and "always" operators can be defined as $\mathrm{F}\gamma \equiv \top \,\mathrm{U}\, \gamma$ and $\mathrm{G}\gamma \equiv \neg\mathrm{F}\neg\gamma$. 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 $\gamma$. 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 *strategy* for agent $a$ is a function $s_a : St \to Act$ such that $s_a(q) \in d_a(q)$.[5] A *collective strategy* for a group of agents $A = \{a_1, \dots, a_i\}$ is simply a tuple of individual strategies $s_A = \langle s_{a_1}, \dots, s_{a_i} \rangle$. 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. Finally, for a path $\lambda \in St^\omega$, we use $\lambda[i]$ to denote the $i$th state on $\lambda$, and $\lambda[i..\infty]$ to denote the $i$th suffix of $\lambda$. The semantics $\models_{ATL}$ of alternating-time logic is defined below:

$M, q \models \mathsf{p}$ iff $q \in PV(\mathsf{p})$, for $\mathsf{p} \in AP$;
$M, q \models \neg\varphi$ iff $M, q \not\models \varphi$;
$M, q \models \varphi_1 \wedge \varphi_2$ iff $M, q \models \varphi_1$ and $M, q \models \varphi_2$;

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

$M, \lambda \models \varphi$ iff $M, \lambda[0] \models \varphi$;
$M, \lambda \models \neg\gamma$ iff $M, \lambda \not\models \gamma$;
$M, \lambda \models \gamma_1 \wedge \gamma_2$ iff $M, \lambda \models \gamma_1$ and $M, \lambda \models \gamma_2$;
$M, \lambda \models \mathrm{X}\gamma$ iff $M, \lambda[1, \infty] \models \gamma$; and
$M, \lambda \models \gamma_1 \,\mathrm{U}\, \gamma_2$ iff there is an $i \in \mathbb{N}_0$ such that $M, \lambda[i, \infty] \models \gamma_2$ and $M, \lambda[j, \infty] \models \gamma_1$ for all $0 \le j < i$.

*Example 3.* Consider the CGS from Figure 1. Clearly, $M_{caf} \not\models \langle\!\langle br \rangle\!\rangle \mathrm{F}\mathsf{white}$: the brewer cannot provide the customer with white coffee on its own. In fact, even both coffee agents together cannot guarantee that, since the customer may never order white coffee: $M_{caf} \not\models \langle\!\langle br, milky \rangle\!\rangle \mathrm{F}\mathsf{white}$. On the other hand, they can produce black coffee regardless of what the customer asks for: $M_{caf} \models \langle\!\langle br, milky \rangle\!\rangle \mathrm{F}\mathsf{black}$. Finally, they can deprive the customer of coffee if they consistently ignore her requests: $M_{caf} \models \langle\!\langle br, milky \rangle\!\rangle \mathrm{G}(\neg\mathsf{black} \wedge \neg\mathsf{white})$.

**Embedding CTL* in ATL*.** The path quantifiers of CTL* can be expressed in the standard semantics of ATL* as follows [2]: $\mathsf{A}\gamma \equiv \langle\!\langle \emptyset \rangle\!\rangle\gamma$ and $\mathsf{E}\gamma \equiv \langle\!\langle \mathbb{A}\mathrm{gt} \rangle\!\rangle\gamma$. We point out that the above translation of $\mathsf{E}$ does *not* work for several extensions of ATL*, e.g., with imperfect information, nondeterministic strategies, and irrevocable strategies. On the other hand, the translation of $\mathsf{A}$ 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 $atl(\varphi)$ from CTL* to ATL* as follows. First, we convert $\varphi$ so that it only includes universal path quantifiers, and then replace every occurrence of $\mathsf{A}$ with $\langle\!\langle \emptyset \rangle\!\rangle$. For example, $atl(\mathsf{EG}(\mathsf{p}_1 \wedge \mathsf{AF}\mathsf{p}_2)) = \neg\langle\!\langle \emptyset \rangle\!\rangle \mathrm{F}(\neg\mathsf{p}_1 \vee \neg\langle\!\langle \emptyset \rangle\!\rangle \mathrm{F}\mathsf{p}_2)$. Note that if $\varphi$ is a CTL formula then $atl(\varphi)$ is a formula of ATL. By a slight abuse of notation, we will use path quantifiers $\mathsf{A}, \mathsf{E}$ in ATL formulae whenever it is convenient.

[5] Unlike in the original semantics of ATL* [2], we use *memoryless* rather than *perfect recall* strategies. It is well known, however, that the semantics based on the two notions of strategy coincide for all formulae of ATL, cf. [2, 27].

### 2.5 Module Checking of ATL* Specifications

ATL* module checking has been proposed and studied in [18]. The problem can be defined by the straightforward combination of our generic treatment of module checking from Section 2.3 and the semantics of ATL* presented in Section 2.4.

*Example 4.* Consider the multi-agent coffee machine $M_{caf}$ from Example 1. Clearly, $M_{caf} \not\models^r \langle\langle br, milky\rangle\rangle \mathsf{F}\mathsf{white}$ because the environment can keep requesting black coffee. On the other hand, $M_{caf} \models^r \langle\langle br, milky\rangle\rangle \mathsf{F}\mathsf{black}$: the agents can provide the user with black coffee whatever she requests. They can also deprive the user of coffee completely – in fact, the brewer alone can do it by consistently ignoring her requests: $M_{caf} \models^r \langle\langle br\rangle\rangle \mathrm{G}(\neg\mathsf{black} \wedge \neg\mathsf{white})$.

The above formulae can be also used for *model* checking, and they would actually generate the same answers. So, what's the benefit of *module* checking? In module checking, we can *condition the property to be achieved on the behavior of the environment.* For instance, users who never order white coffee can be served by the brewer alone: $M_{caf} \models^r \mathsf{AG}\neg\mathsf{reqw} \rightarrow \langle\langle br\rangle\rangle \mathrm{F}\mathsf{black}$. Note that the same formula in model checking trivially holds since $M_{caf} \not\models \mathsf{AG}\neg\mathsf{reqw}$. Likewise, we have $M_{caf} \models \mathsf{AG}\neg\mathsf{reqb} \rightarrow \langle\langle br\rangle\rangle \mathrm{F}\mathsf{white}$, whereas module checking gives a different and more intuitive answer: $M_{caf} \not\models^r \mathsf{AG}\neg\mathsf{reqb} \rightarrow \langle\langle br\rangle\rangle \mathrm{F}\mathsf{white}$. That is, the brewer cannot handle requests for white coffee on its own, even if the user never orders anything else.

## 3 Imperfect Information

In Section 2, we summarized the main developments in module checking for multi-agent systems with perfect information. That is, we implicitly assumed that both the system and the environment always know the precise global state of the computation. The framework was extended to handle uncertain environments in [21] (for temporal logic specifications) and [18] (for specifications of strategic ability). In this paper, we revise and extend our previous work from [18]. The novel contribution is threefold. First, we give a new interpretation of ATL module checking for uncertain environments (Section 3.1). The one proposed in [18], while mathematically sound, arguably does not capture the feasible patterns of behavior that can be produced by uncertain environments. Secondly, we generalize the problem to modules that include uncertainty also on the part of the system (Section 3.2). Thirdly, we investigate formal properties of the new problem, in terms of expressive power (Section 4) as well as algorithms and computational complexity (Section 5).

### 3.1 Handling Environments with Imperfect Information

So far, we have only considered multi-agent modules in which the environment has complete information about the state of the system. In many practical scenarios this is not the case. Usually, the agents have some private knowledge that the environment cannot access. As an example, think of the coffee machine from Example 1. A realistic model of the machine should include some internal variables that the environment (i.e., the customer) is not supposed to know during the interaction, such as the amount of milk in the container or the amount of coins available for giving change. States that differ only in such hidden information are indistinguishable to the environment. While interaction with an "omniscient" environment corresponds to an arbitrary pruning of transitions in the module, in case of imperfect information the pruning must coincide whenever two computations look the same to the environment.

To handle such scenarios, the definition of multi-agent modules was extended as follows [18].

**Definition 5 (Multi-agent module with uncertain environment).** *A* multi-agent module with uncertain environment *is a multi-agent module further equipped with an indistinguishability relation $\sim_e \subseteq St \times St$ that encodes uncertainty of the environment. We assume $\sim_e$ to be an equivalence.*

We will additionally require that the available choices of the environment are consistent with its indistinguishability relation.

**Definition 6 (Uniformity of modules).** *A multi-agent module with uncertain environment is* uniform *wrt relation $\sim_e$ iff $q \sim_e q'$ implies $d_e(q) = d_e(q')$.*

In [18], we assumed that an uncertain environment can only prune whole subtrees of the execution tree, and when it does, it must do it uniformly. This was arguably a very rough treatment of how the environment can choose to behave. We propose a more subtle treatment below.

Let $M$ be a uniform multi-agent module with uncertain environment. First, we extend the indistinguishability relation to the nodes in the computation tree of $M$. Formally, two nodes $v$ and $v'$ in $tree(M)$ are indistinguishable ($v \cong v'$) iff (1) the length of $v, v'$ in $tree(M)$ is the same, and (2) for each $i$, we have $v[i] \sim_e v'[i]$. Secondly, we will only consider prunings of $tree(M)$ that are imperfect information-consistent. Formally, $T \in tree(M)$ is imperfect information-consistent iff it is uniform wrt $\cong$. We denote the set of such prunings by $exec^i(M)$. Clearly, $exec^i(M) \subseteq exec(M)$.

The module checking problem for uncertain environments is defined analogously to the perfect information case:

**Definition 7 (Module checking for uncertain environments).** *Given a multi-agent module with uncertain environment $M$ and a formula $\varphi$ of logic $\mathcal{L}$, the corresponding module checking problem is defined by the following clause:*

$$M \models^{r,i}_{\mathcal{L}} \varphi \quad \textit{iff} \quad T \models_{\mathcal{L}} \varphi \textit{ for every } T \in exec^i(M).$$

*Example 5.* Consider an extension of the multi-agent coffee machine from Example 1, where the environment can choose to reset the machine while it is preparing coffee. If the machine is reset after the coffee is poured but before it is served, the system proceeds to the error state. Moreover, pressing reset in

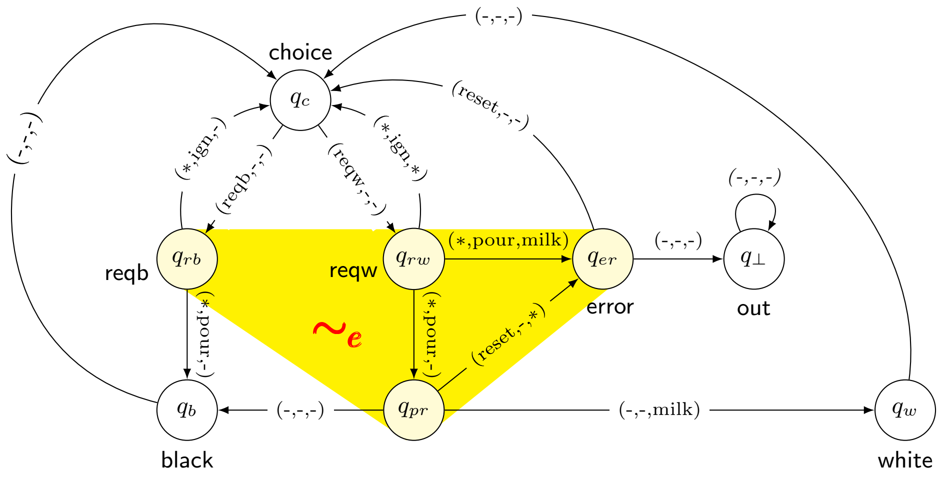

**Fig. 2.** Multi-agent coffee machine with reset $M_{caf2}$

the error state initiates a recovery procedure that brings the system back to the initial state $q_c$. On the other hand, if the system is not reset while in the error state then it proceeds to the "out of order" state $q_{out}$, and requires intervention of an external repair crew. Furthermore, we assume that the environment has no access to the local states of the system agents $br, milky$. Since states $q_{rb}, q_{rw}, q_{pr}, q_{er}$ should intuitively differ only in local states of those agents, they are indistinguishable to the environment (see Figure 2). Note that we do not label states grey and white anymore, as the module is not alternating.

Let us define the "recovery formula" as $\varphi_{recv} \equiv \mathsf{AG}(\mathsf{error} \rightarrow \mathsf{AXchoice})$, saying that the system always recovers after an error. Now we have for example that $M_{caf2} \models^{r,i} \varphi_{recv} \rightarrow \mathsf{AG}\neg\mathsf{white}$. This is because the user cannot distinguish between situations when an error has occurred, and ones where the coffee has been poured and waits for milk to be added. If she chooses to reset the machine in the first kind of nodes, she has to do reset also in the latter, and then white coffee can never be completed. Thus, for such behaviors of the user, the agents cannot provide her with white coffee anymore: $M_{caf2} \models^{r,i} \varphi_{recv} \rightarrow \neg\langle\langle br, milky\rangle\rangle\mathsf{Fwhite}$.

On the other hand, the agents retain the ability to serve black coffee whenever it is requested – in fact, the brewer can make it on its own: $M_{caf2} \models^{r,i} \varphi_{recv} \rightarrow \mathsf{AG}(\mathsf{reqb} \rightarrow \langle\langle br\rangle\rangle\mathsf{Fblack})$. Moreover, for such inputs, the agents cannot crash the system: $M_{caf2} \models^{r,i} \varphi_{recv} \rightarrow \neg\langle\langle br, milky\rangle\rangle\mathsf{Fout}$, which is rather a good thing. Finally, even if the user never tries recovery, the agents can keep the system from crashing, and serve white coffee whenever it is requested (they simply avoid pouring coffee and milk at the same time). Formally, let $\varphi_{norecv} \equiv \mathsf{AG}(\mathsf{error} \rightarrow$

$\mathsf{AX}\neg\mathsf{choice}$); then, $M_{caf2} \models^{r,i} \varphi_{norecv} \to \langle\langle br, milky\rangle\rangle \mathrm{G}\neg\mathsf{out}$ and $M_{caf2} \models^{r,i} \varphi_{norecv} \to \mathsf{AG}(\mathsf{reqw} \to \langle\langle br, milky\rangle\rangle \mathrm{F}\mathsf{white})$.

## 3.2 Imperfect Information Module Checking

The treatment of module checking, presented in the previous section, allows for uncertainty on the part of the environment, but assumes perfect information on the part of the system. That is, the agents that comprise the system can always fully observe the global state of the system, including each other's variables as well as the local state of the environment. Is this assumption realistic? Clearly not. One can perhaps use perfect information models when the hidden information is irrelevant for the agent's decision making, i.e., the agents need only their local views to choose their course of action (cf. the coffee machine example) but even that is seldom justified.

**Definition 8 (Multi-agent module with imperfect information).** *A* multi-agent module with imperfect information *is a multi-agent module further equipped with indistinguishability relations* $\sim_a \subseteq St \times St$, *one per agent* $a \in \mathbb{A}\mathrm{gt}$.

*Each multi-agent module with imperfect information* $M$ *is required to be uniform wrt every relation* $\sim_a$ *in* $M$.

Now we proceed analogously to Section 3.1. Let $M$ be a multi-agent module with imperfect information, and $\mathcal{L}$ be a suitable logic. Two nodes $v$ and $v'$ in $tree(M)$ are indistinguishable to the environment ($v \cong_e v'$) iff (1) the length of $v, v'$ in $tree(M)$ is the same, and (2) for each $i$, we have $v[i] \sim_e v'[i]$. Then, $exec^i(M)$ consists of all the prunings in $exec(M)$ that are uniform wrt $\cong_e$. The corresponding module checking problem is again defined by the clause:

$$M \models^{r,i}_{\mathcal{L}} \varphi \quad \text{iff} \quad T \models_{\mathcal{L}} \varphi \text{ for every } T \in exec^i(M).$$

One thing remains to be settled. What logic is suitable for specification of agents with imperfect information? In this paper, we use a semantic variant of ATL* proposed in [27]. First, a (memoryless) strategy $s_a$ is *uniform* iff $q \sim_a q'$ implies $s_a(q) = s_a(q')$. A collective strategy $s_A$ is uniform iff it consists of uniform individual strategies. Then, the semantics $\models_{ATL_i}$ of "ATL* with imperfect information" is obtained by replacing the clause for $\langle\langle A\rangle\rangle\gamma$ as follows:

$M, q \models \langle\langle A\rangle\rangle\gamma$ iff there is a uniform collective strategy $s_A$ such that, for every $a \in A$, every $q'$ with $q \sim_a q'$, and every $\lambda \in out(q', s_A)$, we have $M, \lambda \models \gamma$.

*Example 6.* Let us go back to the multi-agent coffee machine with reset from Example 5. We will now additionally assume that *milky* cannot detect the *pour* action of the brewer, formally: $q_{rw} \sim_{milky} q_{pr}$. Let us denote the resulting multi-agent model by $M_{caf3}$. Then, the agents are still able to keep the machine from crashing, even for users that do no recovery, but they are not able anymore to guarantee that white coffee requests are served. Formally, $M_{caf3} \models^{r,i} \varphi_{norecv} \to \langle\langle br, milky\rangle\rangle \mathrm{G}\neg\mathsf{out}$ (the right strategy assumes that *milky* never pours milk), and $M_{caf3} \not\models^{r,i} \varphi_{norecv} \to \mathsf{AG}(\mathsf{reqw} \to \langle\langle br, milky\rangle\rangle \mathrm{F}\mathsf{white})$ (in a uniform strategy, if *milky* decides to do no action at $q_{rw}$, it has to do the same at $q_{pr}$).

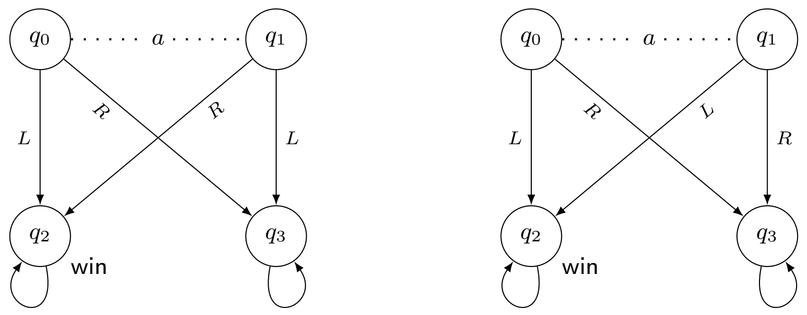

**Fig. 3.** Variants of the "poor duck problem" from [8]

## 4 Expressive Power of Imperfect Information Module Checking

In this section, we show that ATL module checking offers a distinctly different perspective when imperfect information is added. Before we proceed, we briefly recall the notions of distinguishing power and expressive power (cf. e.g. [29]).

**Definition 9 (Distinguishing and expressive power).** *Let $L_1 = (\mathcal{L}_1, \models_1)$ and $L_2 = (\mathcal{L}_2, \models_2)$ be two logical systems 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 $\phi$ 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 $\phi$ 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 $[\![\phi_1, M]\!]_{\models_1} = [\![\phi_2, M]\!]_{\models_2}$.*[6]

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 ATL* can be seen as a logical system where the syntax is given by the syntax of ATL*, and the semantics is given by $\models^r$. For module checking "ATL* with imperfect information", the semantics is given by $\models^{r,i}$. Thus, we can use Definition 9 to compare the expressivity of both decision problems.

**Theorem 1.** *The logical system* (ATL*, $\models^{r,i}$) *has incomparable distinguishing power (and thus also incomparable expressive power) to $(ATL*, \models^r)$.*

*Proof.* First we prove that there are multi-agent modules $M, M'$ that satisfy the same formulae of ATL* wrt the semantic relation $\models^r$, but are distinguished by

[6] 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.

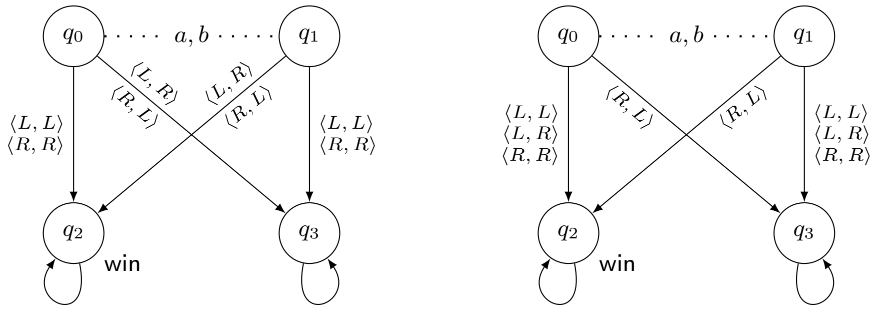

**Fig. 4.** "Coordinated poor duck problem" with 2 agents $a$ and $b$

an ATL* formula wrt the semantic relation $\models^{r,i}$. As $M$, take the "poor duck" model from Figure 3 (left) with $q_0$ be the initial state, and add the environment agent $e$ in such a way that it never influences the evolution of the system (i.e., $|d_e(q)| = 1$ for all $q \in St$). Moreover, let $M'$ be a modified variant of $M$ where the outgoing transitions from $q_1$ are swapped, see Figure 3 (right). Clearly, $exec(M) = exec^i(M) = \{T(M)\}$, and analogously for $M'$ (there is only one way how the environment can act). Thus, the semantic relation $\models^r$ (resp. $\models^{r,i}$) coincides on $M$ and $M'$ with $\models_{ATL}$ (resp. $\models_{ATL_i}$). Furthermore, $M, M'$ are in strategic bisimulation [1], and hence they satisfy the same formulae of ATL* wrt $\models_{ATL}$. On the other hand, $M \not\models_{ATL_i} \langle\langle a\rangle\rangle \mathsf{F}\mathsf{win}$ and $M' \models_{ATL_i} \langle\langle a\rangle\rangle \mathsf{F}\mathsf{win}$.

Secondly, we prove that there are multi-agent modules $M, M'$ that satisfy the same formulae of ATL* wrt the semantic relation $\models^{r,i}$, but are distinguished by an ATL* formula with the semantic relation $\models^r$. As $M$ and $M'$, take now the left hand side and right hand side models from Figure 4, respectively. Again, the initial is $q_0$ and the environment is idle in all states. We leave it for the reader to check that in both models all the coalitions can only enforce trivial path properties (i.e., ones that hold on all paths starting from $q_0, q_1$) by using uniform strategies. Thus, $M$ and $M'$ satisfy the same formulae of ATL$_i$*. On the other hand, $M \not\models_{ATL} \langle\langle a\rangle\rangle \mathsf{F}\mathsf{win}$ and $M' \models_{ATL} \langle\langle a\rangle\rangle \mathsf{F}\mathsf{win}$.

## 5 Algorithms and Complexity

Our algorithmic solution to the problem of ATL* module checking with imperfect information exploits the automata-theoretic approach. It combines and extends that ones used to solve the *CTL* module checking with imperfect information* and the *ATL* model checking with perfect information* problems. Precisely, we make use of *alternating parity tree automata* on infinite tress and reduce the addressed decision problem to the checking for automata emptiness. In this section we first introduce some preliminary definition regarding these automata and then we show how to use them to our purpose. For the sake of clarity we also give a proper definition of infinite labeled trees.

Let $\Upsilon$ be a set. An *$\Upsilon$-tree* is a prefix closed subset $\mathcal{T} \subseteq \Upsilon^*$. The elements of $\mathcal{T}$ are called *nodes* and the empty word $\varepsilon$ is the *root* of $\mathcal{T}$. For $v \in \mathcal{T}$, the set of *children* of $v$ (in $\mathcal{T}$) is $child(\mathcal{T}, v) = \{v \cdot x \in \mathcal{T} \mid x \in \Upsilon\}$. For $v \in T$, a (full) path $\pi$ of $\mathcal{T}$ from $v$ is a *minimal* set $\pi \subseteq \mathcal{T}$ such that $v \in \pi$ and for each $v' \in \pi$ such that $child(\mathcal{T}, v') \neq \emptyset$, there is exactly one node in $child(\mathcal{T}, v')$ belonging to $\pi$. Note that every infinite word $w \in \Upsilon^\omega$ can be thought of as an infinite path in the tree $\Upsilon^*$, namely the path containing all the finite prefixes of $w$. For an alphabet $\Sigma$, a $\Sigma$-labeled $\Upsilon$-tree is a pair $T = \langle \mathcal{T}, V \rangle$ where $\mathcal{T}$ is an $\Upsilon$−tree and $V : \mathcal{T} \to \Sigma$ maps each node of $\mathcal{T}$ to a symbol in $\Sigma$.

In nondeterministic tree automata, on reading a node of the input tree, it is possible to send at most one copy of the automaton in every single child, in accordance with the nondeterministic transition relation. Alternating tree automata, instead, are able to send several copies of the automaton along the same child, by means of a transition relation that uses positive Boolean combinations of directions and states. The formal definition of alternating tree automata follows. For more details we refer to [28, 12].

**Definition 10.** *An alternating tree automaton (*ATA*, for short) is a tuple $\mathcal{A} = < \Sigma,\ D,\ Q,\ q_0,\ \delta,\ F >$, where $\Sigma$ is the alphabet, $D$ is a finite set of directions, $Q$ is the set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$ is the transition function associating to each state and alphabet symbol a positive Boolean combination of pairs $(d, q)$, where $d$ is a direction and $q$ is a state, and $F$ is the accepting condition defined later.*

To give an intuition on how an *ATA* $\mathcal{A}$ works, assume that it is in a state $q$, reading a node tree labeled by $\sigma$ and $\delta(q, \sigma) = ((0, q_1) \vee (1, q_2)) \wedge (1, q_1)$. Then the automaton can just send two copies in direction 1 with state $q_1$ and $q_2$, respectively. The connectives $\vee$ and $\wedge$ in $\delta$ represent, respectively, choice and concurrency. Nondeterministic tree automata are alternating tree automata in which the concurrency feature is not allowed. A run of an alternating tree automaton $\mathcal{A}$ on a $\Sigma$-labeled tree $< \mathcal{T}, V >$, with $\mathcal{T} = D^*$, is a $(D^* \times Q)$-labeled $\mathbb{N}$-tree $< \mathcal{T}_r, r >$ such that the root is labeled with $(\varepsilon, q_0)$ and the labels of each node and its successors satisfy the transition relation. A run $\langle \mathcal{T}_r, r \rangle$ is accepting iff all its infinite paths satisfy the acceptance condition. In this paper we are interested in the parity acceptance condition and, as its special case, the Büchi acceptance condition. A parity condition $F$ maps all states of the automaton to a finite set of colors $C = \{C_{\min}, \ldots, C_{\max}\} \subset \mathbb{N}$. Thus, $F : Q \to C$. For a path $\pi$, let $m(\pi)$ be the maximal color that appears infinitely often along $\pi$. Then, $\pi$ satisfies the parity condition $F$ iff $m(\pi)$ is even. The Büchi acceptance condition is a parity condition with only two colors, i.e., $C = \{1, 2\}$. By $L(\mathcal{A})$ we denote the set of trees accepted by $\mathcal{A}$. We say that the automaton is not empty if $L(\mathcal{A}) \neq \emptyset$. We name ATA along with the parity and Büchi conditions PATA and BATA for short, respectively. In both cases of PATA and BATA emptiness can be checked in EXPTIME [12].

In ATL* module checking with (im)perfect information given a module $M$ and an ATL* formula $\varphi$ we check whether $M \models^{r,(i)} \varphi$ by checking whether

$T \models \varphi$ for every $T \in exec^{(i)}(M)$. Consequently, $M \not\models^{r,(i)} \varphi$ iff there exists a tree $\overline{T} \in exec^{i}(M)$ such that $\overline{T} \models \neg\varphi$. In the perfect information case, to solve this problem one can build a PATA $\mathcal{A}$, accepting all such $\overline{T}$ trees and reduce the model checking question "does $M \models^{r} \varphi$ ?" by checking for the automaton emptiness. In particular, the automaton uses one direction for each possible decision and, input trees and run trees have exactly the same shape. In the case of imperfect information, we are forced to restrict our reasoning to uniform strategies and this deeply complicates the construction of the automaton. Indeed, not all the trees in $exec^{i}(M)$ can be taken into consideration but only those coming from uniform strategies. This has to be taken into account both on the side of the environment agent, while performing the pruning, and the other players playing in accordance with the modalities indicated by the ATL* formula. Uniformity forces to use the same action in indistinguishable states. To accomplish this, the automaton takes as input not trees $T$ from $exec^{i}(M)$, but rather corresponding "thin" trees $T'$ such that each node $v'$ in $T'$ is meant to represent all nodes $H$ in $T$ that are indistinguishable to $v'$. Then, the automaton will send to $v$ just $|H|$ different states all with the same direction, to force all of them to respect the same strategy. Thus, the input tree can be seen as a profile of uniform strategies, e.g., once a uniform strategy has been fixed, it collects all the possible outcomes obtained by combining this strategy with all possible uniform strategies coming from the other players. It is worth noting that the run tree has the shape of the desired $\overline{T} \in exec^{i}(M)$. In a way this is the witness of our automata approach.

To give few more details, let $[St]_{\sim i}$ be the equivalence class build upon the states that are indistinguishable to agent $i$. We use as directions of the automaton $\Pi_i [St]_{\sim i}$. Agents can then choose actions upon their visibility. The automaton has to accept trees corresponding to uniform strategy profiles whose composition with the module satisfy $\neg\varphi$. Thus, a run of the automaton proceeds by simulating an unwinding of the module, pruned at each step according to the strategy profile and the satisfiability of the formula is checked on the fly. Starting from an ATL* formula the automaton we obtain is an exponential PATA. In case of ATL, the automaton is a polynomial BATA. Since the module-checking problem with imperfect information is 2EXPTIME-complete for CTL* and EXPTIME-complete for CTL even in case the formula is of bounded size, we get the following result.

**Theorem 2.** *The module-checking problem with imperfect information is* 2EXPTIME-*complete for ATL* and* EXPTIME-*complete for ATL. For formulae of bounded size the problem is* EXPTIME-*complete in both cases.*

## 6 Conclusions

We have presented an extension of the module checking problem that handles specifications of strategic ability for modules involving imperfect information. As usual for computational problems, the key features are expressivity and complexity. We show that this new variant of module checking fares well in both respects. On one hand, the computational complexity is the same as that of module checking CTL/CTL* with imperfect information. On the other hand, ATL/ATL* module checking under imperfect information has incomparable expressive power to ATL/ATL* module checking for perfect information, which means that the two variants of the problem offer distinctly different perspectives at verification of open systems.

In the future, we plan to characterize the correspondence of imperfect information module checking to an appropriate variant of model checking (in the spirit of [17]). We are also going to look at the relation of module checking to model checking of temporal logics with propositional quantification [19, 23]. Last but not least, we would like to apply the framework to verification of agent-oriented programs.

**Acknowledgements.** Aniello Murano acknowledges the support of the FP7 EU project 600958-SHERPA. Wojciech Jamroga acknowledges the support of the FP7 EU project ReVINK (PIEF-GA-2012-626398).

## References

1. T. Ågotnes, V. Goranko, and W. Jamroga. Alternating-time temporal logics with irrevocable strategies. In *Proceedings of TARK XI*, pages 15–24, 2007.
2. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.
3. 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.
4. B. Aminof, A. Murano, and M. Vardi. Pushdown module checking with imperfect information. In *Proceedings of CONCUR*, LNCS 4703, pages 461–476. Springer-Verlag, 2007.
5. 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.
6. L. Bozzelli. New results on pushdown module checking with imperfect information. In *Proceedings of GandALF*, volume 54 of *EPTCS*, pages 162–177, 2011.
7. L. Bozzelli, A. Murano, and A. Peron. Pushdown module checking. *Formal Methods in System Design*, 36(1):65–95, 2010.
8. N. Bulling and W. Jamroga. Comparing variants of strategic ability: How uncertainty and memory influence general properties of games. *Journal of Autonomous Agents and Multi-Agent Systems*, 28(3):474–518, 2014.
9. E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of Logics of Programs Workshop*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, 1981.
10. 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.
11. E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, 1990.
12. E. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 368–377. IEEE, 1991.

13. A. Ferrante, A. Murano, and M. Parente. Enriched $\mu$-calculi module checking. *Logical Methods in Computer Science*, 4(3:1):1–21, 2008.
14. M. Gesell and K. Schneider. Modular verification of synchronous programs. In *Proceedings of ACSD*, pages 70–79. IEEE, 2013.
15. P. Godefroid. Reasoning about abstract open systems with generalized module checking. In *Proceedings of EMSOFT*, volume 2855 of *LNCS*, pages 223–240. Springer, 2003.
16. 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.
17. W. Jamroga and A. Murano. On module checking and strategies. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2014*, pages 701–708, 2014.
18. W. Jamroga and A. Murano. Module checking of strategic ability. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2015*, pages 227–235, 2015.
19. O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. *Journal of Logic and Computation*, 9(2):135–147, 1999.
20. O. Kupferman and M. Vardi. Module checking. In *Procedings of CAV*, volume 1102 of *LNCS*, pages 75–86. Springer, 1996.
21. O. Kupferman and M. Vardi. Module checking revisited. In *Proceedings of CAV*, volume 1254 of *LNCS*, pages 36–47. Springer, 1997.
22. O. Kupferman, M. Vardi, and P. Wolper. Module checking. *Information and Computation*, 164(2):322–344, 2001.
23. A. D. C. Lopes, F. Laroussinie, and N. Markey. Quantified CTL: expressiveness and model checking. In *Proceedings of CONCUR*, pages 177–192, 2012.
24. F. Martinelli. Module checking through partial model checking. Technical report, CNR Roma - TR-06, 2002.
25. 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.
26. A. Murano, M. Napoli, and M. Parente. Program complexity in hierarchical module checking. In *Proceedings of LPAR*, volume 5330 of *Lecture Notes in Computer Science*, pages 318–332. Springer, 2008.
27. P. Y. Schobbens. Alternating-time logic with imperfect recall. *Electronic Notes in Theoretical Computer Science*, 85(2):82–93, 2004.
28. W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, 2, 1990.
29. Y. Wang and F. Dechesne. On expressive power and class invariance. *CoRR*, abs/0905.4332, 2009.