Synthesis and Verification of Uniform Strategies for Multi-Agent Systems

Jerzy Pilecki[1], Marek A Bednarczyk[2,3], and Wojciech Jamroga[4,5]

[1] Systems Research Institute, Polish Academy of Sciences
[2] Polish-Japanese Institute of Information Technology
[3] Institute of Computer Science, Polish Academy of Sciences
[4] CSC & SnT, University of Luxembourg
[5] Department of Informatics, Clausthal University of Technology

Abstract. We present a model checking algorithm for alternating-time temporal logic (ATL) with imperfect information and imperfect recall. This variant of ATL is arguably most appropriate when it comes to modeling and specification of multi-agent systems. The related variant of model checking is known to be theoretically hard ($\mathbf{\Delta_2^P}$- to **PSPACE**-complete, depending on the assumptions), but virtually no *practical* attempts at it have been proposed so far. Our algorithm searches through the set of possible uniform strategies, utilizing a simple reduction technique. In consequence, it not only verifies existence of a suitable strategy but also produces one (if it exists). We validate the algorithm experimentally on a simple scalable class of models, with promising results.

Keywords: model checking, alternating-time logic, imperfect information, strategy synthesis

1 Introduction

There is a growing number of works that study syntactic and semantic variants of *strategic logics*, in particular the alternating-time temporal logic ATL. Conceptually, the most interesting strand builds upon reasoning about temporal patterns and outcomes strategic play, limited by information available to the agents. The contributions are mainly theoretical, and include results concerning the conceptual soundness of a given semantics of ability [20, 1, 12], meta-logical properties [7], and the complexity of model checking [20, 11, 10]. However, there is very little research on actual *use* of the logics, in particular on practical algorithms for reasoning and/or verification.

This is somewhat easy to understand, since model checking of ATL variants with imperfect information has been proved $\mathbf{\Delta_2^P}$- to **PSPACE**-complete for agents playing positional (a.k.a. memoryless) strategies [20, 11] and undecidable for agents with perfect recall of the past [9]. Moreover, the imperfect information semantics of ATL does not admit fixpoint equivalences [7], which makes incremental synthesis of strategies impossible, or at least cumbersome. Still, some other results [10, 21] suggest that practical model checking of strategies with imperfect information might not be actually *that* harder than the standard perfect information case, for which successful algorithms and model checkers already exist [6, 3, 13, 17, 16]. Either way, we believe that the scientific approach requires an extensive study of the practical hardness of the problem. This paper is our first step in that direction.

We propose a novel model checking algorithm for a fragment of alternating-time temporal logic with imperfect information and memoryless strategies (ATL$_{ir}$). When model checking a formula of type $\langle\langle a\rangle\rangle\gamma$, the algorithm tries to synthesize an executable (i.e., uniform) strategy for agent a that would enforce property γ. The task requires to search through exponentially many strategies in the worst case; however, we build on some observations that lead to a reduction of the search space for certain instances of the problem. In consequence, a significant decrease in complexity is possible for many practical instances.

Our algorithm comes in two variants: one based on exhaustive search through the space of all uniform strategies, and another one based on a simple constructive heuristic. The latter variant tries to construct the strategy by "blindly" following a single path in the model. We evaluate both variants experimentally on a simple scalable class of models. In terms of comparison to existing results we have faced a difficult problem, since there are virtually no results to compare with. The only existing tool for MAS that verifies existence of executable strategies under imperfect information is an experimental version of MCMAS [19]. We compare the performance of our algorithm to that version, with very promising results. Moreover, some model checkers admit imperfect information *models* but use perfect information (i.e., possibly non-executable) *strategies* in the semantics [3, 17, 16]. We compare the performance of our algorithm to one of those tools (the standard version of MCMAS [16]) in order to get a grip on how imperfect information changes the practical verification complexity. The only other model checking algorithm for ATL$_{ir}$ that we know of [8] has been studied in [18], with results that suggested bad performance.

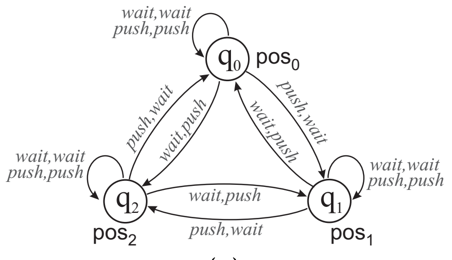
2 Preliminaries

We begin by presenting the syntax and semantics of alternating-time temporal logic, as well as defining the model checking problem formally.

2.1 ATL: What Agents Can Achieve

Alternating-time temporal logic (ATL) was proposed in [4, 5] for reasoning about abilities of agents in multi-agent systems. Intuitively, formula $\langle\langle A\rangle\rangle\varphi$ expresses that the group of agents A has a collective strategy to enforce φ. The formal syntax of ATL is given by the following grammar:

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



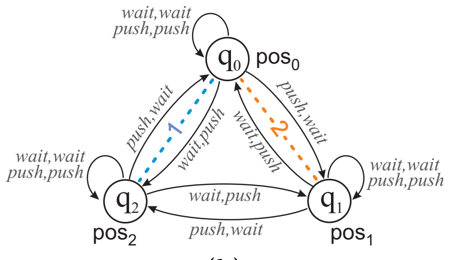


Fig. 1. Robots and carriage: (a) concurrent game structure M_1; (b) iCGS M_2

where p is an atomic proposition, A is a subset of agents, and the operators X, G, and $\mathcal{U}$ stand for "in the next state", "always from now on", and "strong until", respectively. Additional operator F ("eventually") can be defined as $F\,\varphi \equiv \top\,\mathcal{U}\,\varphi$.

ATL is interpreted in a variant of transition systems where transitions are labeled with combinations of actions, one per agent. Formally, a *concurrent game structure* is a tuple $M = \langle \Sigma, Q, \Pi, \pi, d, \delta \rangle$, where: $\Sigma = \{1, \dots, k\}$ is a finite nonempty set of *players* (also called *agents*), Q is a finite nonempty set of *states*, $\pi : Q \to 2^{\Pi}$ is the *labeling function*. Moreover, for each player $a \in \{1, \dots, k\}$ and state $q \in Q$, $d_a(q) \geq 1$ gives the number of moves available to a at q; we identify the moves of a at q with the numbers $1, \dots, d_a(q)$. For each state $q \in Q$, a *move vector* at q is a tuple $\langle j_1, \dots, j_k \rangle$ such that $1 \leq j_a \leq d_a(q)$ for each player a. Furthermore, $D(q)$ denotes the set $\{1, \dots, d_1(q)\} \times \dots \times \{1, \dots, d_k(q)\}$ of move vectors. Finally, δ is the deterministic *transition function* that returns a state $q' = \delta(q, j_1, \dots, j_k)$ for each $q \in Q$ and $\langle j_1, \dots, j_k \rangle \in D(q)$.

The meaning of ATL formulae is based on the notion of a strategy. A *memoryless strategy* for player $a \in \Sigma$ is a function $s_a : Q \to \mathbb{N}$ that maps every state q in the model to an action label $s_a(q) \leq d_a(q)$.[6] A *collective strategy* for agents $A \subseteq \Sigma$ is simply a tuple of strategies, one per agent in A. Each collective strategy S_A induces a set of computations (paths, runs). Formally, by $out(q, S_A)$ we will denote the set of infinite sequences of states that can occur from state q on when the players in A follow strategy S_A and the other players are free to do any actions. The semantic relation for ATL is defined inductively as follows:

- $M, q \models p$, for proposition $p \in \Pi$, iff $p \in \pi(q)$
- $M, q \models \neg\varphi$ iff $M, q \not\models \varphi$
- $M, q \models \varphi_1 \vee \varphi_2$ iff $M, q \models \varphi_1$ or $M, q \models \varphi_2$
- $M, q \models \langle\!\langle A \rangle\!\rangle X\varphi$ iff there exists a collective strategy S_A such that for all computations $\lambda \in out(q, S_A)$, we have $M, \lambda[1] \models \varphi$
- $M, q \models \langle\!\langle A \rangle\!\rangle G\varphi$ iff there exists a collective strategy S_A such that for all computations $\lambda \in out(q, S_A)$, and all positions $i \geq 0$, we have $M, \lambda[i] \models \varphi$.

[6] We depart from the assumption in [4, 5] that agents have perfect recall of past situations. Note that both types of strategies (memoryless and perfect recall) yield equivalent semantics in case of standard ATL [5, 20].

– $M, q \models \langle\!\langle A \rangle\!\rangle \varphi_1 U \varphi_2$ iff there exists S_A such that for all $\lambda \in out(q, S_A)$ there is $i \geq 0$ with $M, \lambda[i] \models \varphi_2$ and for all $0 \leq j < i$ we have $M, \lambda[j] \models \varphi_1$.

Example 1. An example concurrent game structure is depicted in Figure 1a. Some ATL formulae that hold in state q_0 of the model are: $\langle\!\langle 1, 2 \rangle\!\rangle F\, \mathsf{pos}_1$ (robots 1 and 2 have a collective strategy to make the carriage eventually reach position 1), $\neg\langle\!\langle 1 \rangle\!\rangle F\, \mathsf{pos}_1$ (robot 1 cannot bring about it on its own), $\langle\!\langle 1 \rangle\!\rangle G\, \neg\mathsf{pos}_1$ (on the other hand, robot 1 can singlehandedly avoid position 1 forever).

2.2 Abilities under Imperfect Information

The assumption that agents know the entire state of the system at each step of its execution is usually unrealistic; similarly, assuming perfect recall is not always practical [20, 1, 12]. The tension between *perfect* and *imperfect information*, as well as between *perfect* and *imperfect recall*, gives rise to the four "classical" semantic variants of ATL from [20]. On the level of models, we extend concurrent game structures to *imperfect information concurrent game structures (iCGS)* by adding indistinguishability relations $\sim_a \subseteq Q \times Q$, one per $a \in \Sigma$. Intuitively $q \sim_a q'$ iff a cannot distinguish q from q'. Then, *local states* of agent a can be defined as equivalence classes of the indistinguishability relation, denoted $[q]_{\sim_a}$.

In this paper, we are interested in the imperfect information + imperfect recall variant (ATL$_{ir}$), with the following semantics. First, we require strategies to be *uniform*, i.e., to specify the same choices in indistinguishable states; formally: if $q \sim_a q'$ then $s_a(q) = s_a(q')$. This ensures that the choice of an action does not depend on information that is inaccessible to the agent. Secondly, a collective strategy is uniform iff it consists only of uniform individual strategies. Thirdly, we update the semantic clauses from Section 2.1 by requiring all strategies to be uniform. Note that this semantics differs slightly from the one in [20] in that it looks only at the outcome paths starting from the current *objective* state of the system. We refer the interested reader to [7, 2] for the philosophical discussion, and point out that it does not affect our performance results in Section 5, as the models in the experiments have a relatively small number of global states indistinguishable from the objective initial state. Moreover, model checking in the "subjective" semantics from [20] can be easily simulated in our "objective" semantics by having the environment agent inject nondeterminism on the first transition. We omit further details for lack of space.

Example 2. An example iCGS is depicted in Figure 1b. Now, formula $\langle\!\langle 1 \rangle\!\rangle G\, \neg\mathsf{pos}_1$ does not hold in q_0 anymore: in order to avoid state q_1, robot 1 should wait in q_0 and push in q_1, which is not allowed in a uniform strategy.

2.3 Model Checking Problem

The decision problem of *local model checking* is typically defined as follows. Given a model M, an initial state q in the model, and a formula φ, determine whether $M, q \models \varphi$. Model checking of ATL with perfect information is known to be linear wrt the length of the formula and the number of global transitions in the model [4, 5]. Model checking of ATL$_{ir}$ is much harder, namely $\mathbf{\Delta_2^P}$-complete [20, 11]. Moreover, for formulae with a single non-negated coalitional modality it becomes **NP**-complete [20]. This is mainly because fixpoint characterizations of strategic modalities do *not* hold under imperfect information [7], and hence purely incremental synthesis of winning strategies is not possible for ATL$_{ir}$.

3 Towards ATL$_{ir}$ Model Checking

As the starting point of our approach, we take the simple nondeterministic algorithm from [20] that model-checks formula $\langle\langle A\rangle\rangle\varphi$ in M, q:

1. Guess nondeterministically a collective uniform strategy S_A;
2. Perform CTL model checking of $\mathsf{A}\varphi$ ("for all paths φ") in $M \dagger S_A, q$, where $M \dagger S_A$ denotes model M "trimmed" according to strategy S_A.

For nested strategic modalities, the algorithm proceeds recursively (bottom-up).

In order to construct a working version of the algorithm, we need to determine the order in which the space of solutions (i.e., strategies) will be searched. The key to such determinization is a heuristic. With a good heuristic, we can hope to achieve acceptable computation time at least for instances where a solution exists. This has been experimentally observed for several classes of computationally hard problems, most notably SAT. Our heuristic is based on three factors. First, we reduce the search space by exploring some equivalences between strategies. Secondly, we define a representation of strategies that minimizes the cost of storing and processing a strategy, but even more importantly makes the algorithm try simpler solutions first. Thirdly, we define a subclass of strategies that are relatively simple to construct and verify – which yields an incomplete but reasonably efficient variant of the model checking algorithm. We present the ideas in detail in the remainder of this section.

3.1 Restricting the Search Space

In case of ATL$_{ir}$ model checking, the solutions are strategies that a coalition can use to enforce a property.[7] Since the space of solutions is computationally large, it is crucial for the algorithm to limit the search space as much as possible. We limit the search space by identifying some equivalences between solutions.

Definition 1. *For a model M and strategy S for coalition A, we define a* trimmed model *M_S as a restriction of model M, where agents from coalition A have their choices restricted by S. $Q_S \subseteq Q$ will denote the set of states reachable in M_S. We will call Q_S the* proper domain *of strategy S in model M.*

We also consider strategies that are not completely specified.

[7] From now on, when referring to strategies, we mean *uniform memoryless strategies*.

Definition 2. *An* incomplete strategy *is a strategy represented by a partial rather than total function, i.e., $s : Q \rightharpoonup \mathbb{N}$. As usual, the* domain *of s ($dom(s)$) is the subset of Q where the value of s is defined. The definitions of trimmed model and proper domain can be easily extended to incomplete strategies.*

In the naive approach, we can take the domain of a strategy to be the whole Q. Note, however, that the assignment of actions for states in $Q \setminus Q_S$ does not have any significance, because those states are never reached with strategy S. We observe that strategies S_1, S_2 that assign identical actions in the same *proper domain* $Q_{S_1} = Q_{S_2}$ can be considered *equivalent*, regardless of actions assigned in $Q \setminus Q_{S_1}$. The equivalence class can be represented by a partial function which is only defined for the relevant states in Q, i.e., for states in Q_S.

Definition 3. *An incomplete strategy s is* proper *iff $dom(s) = Q_s$.*

Since only proper strategies are worth considering, we can significantly limit the searched strategy space by treating all strategies equivalent to S as a single proper strategy. This single proper strategy can be viewed a representative of an equivalence class of strategies. The size of each such equivalence class can be described as

$$\prod_{a \in A} \prod_{[q]_{\sim_a} \in [Q \setminus Q_S]_{\sim_a}} Act([q]_{\sim_a})$$

where $Act([q]_{\sim_a})$ denotes the number of actions available for agent a in the equivalence class of states $[q]_{\sim_a}$.

3.2 Representation of Partial Strategies

Proper strategies are incomplete in the sense that they leave the actions at unreachable states undefined. Still, in their proper domain, they are completely deterministic. In many cases it is worth considering *partial* strategies that leave some choices open, even in the reachable zone. The intuition is: in some states, all choices work equally well, and thus it is not necessary to fix a deterministic choice in those states.

Definition 4. *A* partial strategy *for agent a in model M is a nondeterministic, possibly incomplete strategy $s_a : Q \rightharpoonup 2^{\mathbb{N}}$ such that, for each $q \in dom(s)$, we have either $s_a(q) = d_a(q)$ or s_a is a singleton.*

The explicit part *of a partial strategy s_a is the part of s_a where $s_a(q)$ is always a singleton. The* implicit part *of a partial strategy s_a is the part of s_a where $s_a(q) = d(q)$. We will refer to the explicit and implicit parts of s_a as $expl(s_a)$ and $impl(s_a)$, respectively. Also, we will sometimes call $dom(expl(s))$ the* explicit domain *of s, and $dom(impl(s))$ the* implicit domain *of s.*

Definition 5. *We define the* size *of a strategy s as the number of indistinguishability classes of states contained in $dom(s)$. A partial strategy s is* empty *iff $expl(s)$ has size 0. Conversely, s is* fully determined *iff $impl(s)$ has size 0.*

In a model M, the move function D determines the sets of actions available to an agent in any state. A partial strategy can be seen as a possible restriction on the function. An empty strategy is just a strategy that imposes no restriction. A fully determined strategy, on the other hand, assigns a concrete action to every relevant state. All other partial strategies have explicit assignments for some states, and implicit for the others (according to the move function D).

Example 3. Consider a model with 2 states $Q = \{color, noColor\}$, with a single agent with 2 actions $Act = \{push, wait\}$. The move function in the model permits the execution of both actions in both states.

An empty strategy ES is equivalent to the move function, i.e. it permits the execution of both actions in both states as well. An example fully determined strategy CS defined in the following way: $CS(x) = \{\text{push if x = color, wait if x = noColor}\}$ assigns a single action for all states in Q_S, leaving the implicit strategy empty (of size 0). An example partial strategy PS defined in the explicit part in the following way: $PS_{explicit}(x) = \{\text{push if x = color}\}$ must have the implicit domain cover the rest of states in Q_S, and therefore the implicit strategy is: $PS_{implicit}(x) = \{\{\text{push, wait}\} \text{ if x = noColor}\}$.

The above concepts are so far only specified for individual strategies. This can be easily extended to coalitional strategies. A partial strategy for $A \subseteq \Sigma$ is simply a tuple of partial strategies for $a \in A$. It is empty iff all its components are empty, fully determined iff all its components are determined, etc.

3.3 Looking for Strategies on a Path

As we will see in Section 5, restricting the search to proper strategies and starting the synthesis from the empty partial strategy brings considerable computational benefits. In many cases, however, the space of potential solutions is still huge. In this section, we propose to consider a strict subclass of strategies that fix deterministic choices on a single path only, and leave choices off the path open. Our ultimate heuristic will be to look at such strategies only, which should work well for models with a limited degree of nondeterminism.

Definition 6. *We call a sequence of states $(q_1, \ldots, q_n)$ a* lasso *in M iff: (a) there is a transition in M between every q_i and q_{i+1}, and (b) there is a transition between q_n and some q_i, $1 \leq i \leq n$. Note that a lasso implicitly defines an infinite path that starts with $(q_1, \ldots, q_n)$ and then cycles in the periodic part.*

We call $(q_1, \ldots, q_n)$ a line *iff condition (a) is satisfied.*

Definition 7. *A partial strategy S is* path-based *iff $dom(expl(S))$ is a lasso in M_S. Moreover, S is* bounded path-based *iff $dom(expl(S))$ is a line in M_S.*

4 The SMC Model Checker

SMC (Strategic Model Checker) is a software tool designed for model checking ATL$_{ir}$ and synthesis of uniform strategies. The current version of SMC can model-check ATL$_{ir}$ formulae that contain at most a single coalitional modality. More precisely, the following formulae classes are supported:

- φ
- $\langle\!\langle A\rangle\!\rangle G\varphi$
- $\langle\!\langle A\rangle\!\rangle F\varphi$
- $\langle\!\langle A\rangle\!\rangle X\varphi$
- $\langle\!\langle A\rangle\!\rangle \varphi U\varphi'$

where φ, φ' are boolean formulae. Extension to the full logic of ATL$_{ir}$ is planned as the next step. We note, however, that the importance of formulae with nested modalities is rather limited. For instance, formula $\langle\!\langle A\rangle\!\rangle F\,\langle\!\langle B\rangle\!\rangle G\,\mathsf{p}$ refer to A'a ability to enable some ability of B – in this case, to maintain p forever. This kind of properties is specified rather seldom; much more often, one wants to make sure that some agents A can bring about a *factual* state of affairs p (e.g., by specifying and verifying formula $\langle\!\langle A\rangle\!\rangle F\,\mathsf{p}$).

In this section, we present the algorithm behind SMC. We start with a general description, then provide a more detailed description of the most important step, and eventually an in-depth description of that step.

4.1 High-Level Description of the Algorithm

The general structure of the algorithm is as follows:

1. For formula of type $\langle\!\langle C\rangle\!\rangle\varphi$, synthesize a previously unverified strategy S_C to be verified;
2. Model-check the CTL formula $\mathsf{A}\varphi$ in the trimmed model $M \dagger S_C$;
3. If step 2 returns *true* then terminate returning *true* together with the strategy S_C;
4. If all strategies have been verified, return *false* and terminate;
5. Else, return to start.

Step 1 (strategy synthesis step) is the most significant, as step 2 can be performed with the well-known fixpoint model checking algorithm for CTL, with a slightly modified pre-image function that operates on iCGS's. Points 3–5 are simple binary decision steps. Thus, our next move is to elaborate on step 1:

1. Start with an empty partial strategy and with the initial state;
2. In a loop, generate potential partial strategies by fixing actions for newly discovered states that do not have already fixed actions. These newly discovered states are required to be reachable with the employment of this strategy;
3. Continue the above step until a successful strategy is found or all strategies have been explored.

4.2 Low-Level Description of Strategy Synthesis

In order to implement the strategy synthesis step, we define the following structures. A *strategy task* $ST = \langle F, U, S \rangle$ consists of:

1. The set of *fixed* states F. For any state in F we have already assigned actions for all agents in the explicit domain of the partial strategy S;
2. The set of *unchecked* states U. States in U may have no explicit actions assigned in S yet for some or all of the agents;
3. The partial strategy S.

A *strategy tasks list* STL is a list of strategy tasks. We will implement STL as a sequential data structure (e.g. queue or stack) that stores the strategy tasks to be processed in the future.
The list is initialized with $STL_0 = \{\langle \emptyset, \{initialState\}, emptyPartialStrategy \rangle\}$.

The strategy synthesis algorithm proceeds as follows:

1. If $STL = \emptyset$, terminate with answer *no strategy found*. Otherwise remove a strategy task from STL in order to process it. This current strategy task will be referred to as $CST = \langle F, U, S \rangle$;
2. Fix a current state $CS \in U$ and do $F = F \cup \{CS\}$, $U = U \setminus \{CS\}$;
3. Generate all possible children strategies for S, reachable by fixing a previously unfixed action for the current state CS. (Note: This step generates strategies if at least one of the agents in the checked coalition has an unfixed action in the current state CS. We do not fix actions for agents that already have a fixed action in this state.)
4. If there were no new strategies generated in step 3, generate a new strategy task $\langle F, U, S \rangle$. (Note: the strategy is still S, but we have changed F and U in step 2.) Add this strategy task to STL if $U \neq \emptyset$. Assume S as current strategy;
5. If there were new strategies generated in step 3, process the first strategy as the current strategy. Postpone processing all *other* strategies except the first one by adding appropriate strategy tasks to STL. Do $U = \emptyset$ if path-centric synthesis is enabled. Add to U the successors (states reachable in a single step) of CS, that are not present in $F \cup U$.
6. If only unbounded (complete) strategies verification is enabled, ignore this step unless the current strategy is an unbounded (respectively, complete) strategy. Otherwise, pass the current strategy to the verification step (done by means of CTL fix-point model checking of the trimmed model $M \dagger S$). If the verification yields *true* result, terminate with answer *strategy found* and return the current strategy as witness;
7. Return to step 1.

In order to ensure that the algorithm is well-understood, some further explanations are needed. As stated in the high-level description, the crucial point is that we extend partial strategies by adding a single entry into the explicit domain of a partial strategy. For any agent in the checked coalition, we add a

single entry that fixes the action in this state, unless such an action is already fixed. The possibility of this action being fixed in a previously unchecked global state stems from the presence of imperfect information. While this global state has certainly not been checked before, it might be indistinguishable for this particular agent with a state that has been checked before. In such a situation this agent has an action for the equivalence class containing both those states already fixed. Step 4 describes a very special case of such an event, where we have a previously unchecked global state that has already fixed actions for all agents in the coalition. Step 5 on the other hand describes a situation where at least one of the agents has no fixed action for this current global state, therefore has the possibility to extend his partial strategy by adding a new entry in the explicit domain of the strategy.
In step 5, if path-centric synthesis is enabled, the algorithm only considers as sources of strategy refinement states reachable from the current state (CS). This is achieved by doing $U = \emptyset$. Essentially, always only one successor of the current state CS is used to extend a strategy, then all other successors are forgotten. This leads to path-centric strategies.

4.3 Discussion

Our approach enables the capability of constructing strategies of limited explicit strategy size. To illustrate the idea that fully determined strategies with a large domain are not always required, we present an example where a partial strategy with the explicit domain size 1 is sufficient.

Example 4. Consider a model of a game of checkers with two players, a and b. The formula is $\langle\langle a \rangle\rangle F \mathsf{playerAHasLessPiecesThanCurrently}$. The meaning of this formula is that agent a has a strategy to enforce himself having in the future at least one piece less than he currently has. The initial state of the model is an already started game where a move for a exists that forces b to capture a piece in the next transition of the system. Therefore there exists a successful partial strategy to satisfy the verified formula where the explicit domain of this strategy has size 1. In other words, a successful partial strategy that just assigns a single action in the initial state is possible to be constructed.

This example demonstrates that it is not necessary to build fully determined nor unbounded strategies sometimes. The output of this example can be a bounded path-based strategy of size 1, as the explicit part suffices as output. On the other hand, generating a fully determined strategy could easily require a domain size of 10^3 or more.

A proper partial strategy can be described by the explicit strategy part. This part can often be of small size, what the example above illustrates. Example benefits of smaller size of strategy domains are improved readability for humans and reduced memory/processing requirements for computers.

4.4 Variants of the Algorithm

In the experiments, we will use three different versions of the SMC algorithm. *SMC with branching strategy search* searches through all the proper strategies, which usually requires fixing choices for multiple successors of a given state (hence the "branching" moniker). *SMC with path-based strategy search* searches only through path-based strategies, and *SMC with bounded path-based strategy search* searches only through bounded path-based strategies.

We call a variant of SMC *sound* iff $SMC(M, q, \langle\langle A\rangle\rangle\varphi) = true$ implies $M, q, \langle\langle A\rangle\rangle \models \varphi$. Conversely, the variant of SMC is *complete* iff $M, q, \langle\langle A\rangle\rangle \models \varphi$ implies $SMC(M, q, \langle\langle A\rangle\rangle\varphi) = true$. The following claims are straightforward:

Theorem 1. *SMC with branching strategy search is sound and complete.*

Theorem 2. *SMC with (bounded or unbounded) path-based strategy search is sound but not necessarily complete.*

5 Experimental Results

In this section, we present experimental results obtained by running the SMC model checker on a parameterized class of models. All the tests have been conducted on a notebook with an Intel Core i7-3630QM CPU with dynamic clock speed of 2.4 GHz up to 3.4 GHz. The clock speed observed in the conducted tests was 3.2 GHz. The computer was equipped with 8 GB of RAM (two modules DDR3 PC3-12800, 800 MHz bus clock, effective data rate 1600 MT/s, in dual-channel configuration). The experiments with SMC were conducted on Windows 7 OS, the experiments with MCMAS on Linux Ubuntu 12.04.2.

5.1 Working Example: Castles

For the experiments, we designed a simple scalable model called *Castles*. The model consists of one agent called *Environment* that keeps track of the health points of three castles, plus a number of agents called *Workers* each of whom works for the benefit of a castle. Health points (HP, ranging from 0 to 3) represent the current condition of the castle; 0 HP means that the castle is *defeated*.

Workers can execute the following actions:

1. attack a castle they do not work for,
2. defend the castle they do work for, or
3. do nothing.

Doing nothing is the only available action to a Worker of a defeated castle. No agent can defend its castle twice in a row, it must wait one step before being able to defend again. A castle gets damaged if the number of attackers is greater than the number of defenders, and the damage is equal to the difference. For example, if castle 3 is attacked by two agents, it loses 2 HP if not defended, or

1 HP if defended by a single agent. In the initial state, all the castles have 3 HP and every Worker can engage in defending its castle.

The indistinguishability relations for Workers are defined as follows. Every Worker knows if it can currently engage in defending its castle, and can observe for each castle if it is defeated or not. This defines 4 observable (boolean) variables for the agent. Now, $q \sim_a q'$ iff q, q' have the same values of the variables.

The model is parameterized by the number of agents and the allocation of Workers. For example, an instance with 1 worker assigned to the first castle, 3 workers assigned to the second and 4 to the third castle will be denoted by 9 $(1, 3, 4)$.

5.2 Performance Results

We begin by presenting some performance results for the formula

$$\varphi_1 \equiv \langle\langle c12 \rangle\rangle F\, \mathsf{castle3Defeated}$$

saying that the agents working for castles 1 and 2 have a collective strategy to defeat castle 3, no matter what the other agents do. Note that the formula is true in all the models that we have tested. We used the SMC variant with (unbounded) path-based strategy search. The timeout was set to 10 minutes.

N	Total time (ms)	1st step (ms)	2nd step (ms)	Peak memory (MB)
4 (1 1 1)	130	100	29	15
5 (1 1 2)	6 686	336	6 349	198
6 (2 1 2)	4 508	548	3 957	606
7 (2 2 2)	3 366	2 637	728	77
8 (3 2 2)	255 549	27 040	228 505	454

The table presents results for a sequence of models of various size. The columns should be interpreted in the following way (from left to right):

1. The scalability factor N: the total number of agents (incl. Environment), followed by the number of agents working for Castles $1, 2, 3$ respectively;
2. Total "wall clock" time taken by the model checking algorithm in milliseconds (excluding the input parsing time);
3. "Wall clock" time taken by the first step of the algorithm (strategy synthesis);
4. "Wall clock" time taken by the second step (CTL verification);
5. Peak memory usage observed during the execution of the program in megabytes.[8]

5.3 Number of Generated Strategies

The table below presents the number of strategies processed by the algorithm, which might be of an even greater interest than raw performance times. The SMC variant, parameters of tests, and the formula are the same as in Section 5.2.

[8] Note that the default Java Virtual Machine makes it hard to determine the real maximum usage, as memory is freed nondeterministically.

N	Agents	Potential strategies	Proper strategies	Tested strategies
4 (1 1 1)	2	$4.3 * 10^8$	283	1
5 (1 1 2)	2	$4.3 * 10^8$	229	4
6 (2 1 2)	3	$8.9 * 10^{12}$	3 507	3
7 (2 2 2)	4	$1.8 * 10^{17}$	$4,4 * 10^5$	1
8 (3 2 2)	5	$3.8 * 10^{21}$	not calculated	3

The columns are interpreted as follows (left to right):

1. The scalability factor N;
2. The number of agents in the coalition for which a strategy is constructed;
3. The total number of *potential* strategies;
4. The total number of *proper unbounded path-based* strategies;
5. The number of strategies processed by the algorithm.

5.4 Comparison to MCMAS

The only tool for ATL$_{ir}$ model checking that we are aware of is an experimental version of MCMAS [19], not yet released publicly at the time of writing this paper. Thanks to the authors of MCMAS who kindly provided us with the experimental version, we could compare the output of both model checkers. All the parameters of the experiments were like in Sections 5.2–5.3, except for the timeout (set to 120 minutes). Moreover, we used the following two formulae:

$$\varphi_1 \equiv \langle\!\langle c12 \rangle\!\rangle F\, \mathsf{castle3Defeated} \quad \text{(same as before; } \textit{true} \text{ in the tested models)}$$

$$\varphi_2 \equiv \langle\!\langle w12 \rangle\!\rangle F\, \mathsf{allDefeated} \quad (\textit{false} \text{ in the tested models})$$

Formula φ_2 says that Workers 1 and 2 have a collective strategy to enforce that all the castles become defeated, no matter what the other agents do. The tables below compare the performance of both model checkers.

N	Formula	MCMAS execution time	SMC execution time
4 (1 1 1)	φ_1	72 s	0.1 s
5 (2 1 1)	φ_1	> 120 mins. (interrupted)	0.2 s
4 (1 1 1)	φ_2	78 s	5.4 s
5 (2 1 1)	φ_2	error	51 s

N	Formula	MCMAS tested strategies	SMC tested strategies
4 (1 1 1)	φ_1	$\approx 20\,000$	1
5 (2 1 1)	φ_1	$> 2 * 10^6$ (interrupted)	1
4 (1 1 1)	φ_2	$\approx 20\,000$	283
5 (2 1 1)	φ_2	error	106

It is important to note that MCMAS and SMC implement slightly different semantics of ATL$_{ir}$. While for SMC a strategy is successful if it succeeds on the paths starting from the actual initial state, MCMAS requires the strategy to succeed also on all the paths starting from indistinguishable states. For coalitional indistinguishability, MCMAS uses the “everybody knows” epistemic relation. A quick calculation shows that the initial epistemic class of a Worker contains $3^3 * 2^{W-1}$ states, where W is the number of Workers in the model. For a coalition of two Workers, there are $27 * 2^{W-1} + 27 * 2^{W-1} - 27 * 2^{W-2} = 81 * 2^{W-2}$ indistinguishable states. Thus, MCMAS needs to check 162 times more paths than SMC for $N = 4(1\ 1\ 1)$, and 324 times more paths for $N = 5(2\ 1\ 1)$.

5.5 Perfect vs. Imperfect Information Strategies

In this work, we also wanted to compare how model checking of abilities under imperfect information compares to the standard ATL case. To this end, we have compared the performance of SMC and the experimental version to the standard version of MCMAS [16]. The table reports model checking times (in milliseconds) for formula φ_1 in various instances of the *Castles* class.

N	perfect info (MCMAS)	imperfect info (SMC)	imperfect info (MCMAS)
4 (1 1 1)	43	130	72 000
5 (1 1 2)	70	6 686	timeout
6 (2 1 2)	250	4 508	timeout
7 (2 2 2)	954	3 366	timeout
8 (3 2 2)	1 996	255 549	timeout

5.6 Path-Based vs. Branching Strategy Search

So far, we have only presented experimental results for the (sound but incomplete) SMC variant using path-based strategy search. Here, we compare its performance to the complete variant, i.e., one that searches all the proper partial strategies. The table below gives the model checking times (in milliseconds) for formula φ_1 in different instances of the class of models.

N	Path-based strategy search	Branching strategy search
4 (1 1 1)	130	769
5 (1 1 2)	6 686	13 630
6 (2 1 2)	4 508	72 419
7 (2 2 2)	3 366	261 704
8 (3 2 2)	255 549	timeout

5.7 Example Output of Strategy Synthesis

One of the most interesting features of SMC is that it not only verifies existence of a suitable strategy, but also returns the strategy. Thus, SMC can be potentially used as a multi-agent planner. To conclude the section, we present some strategies produced by SMC for our working example. We use the model with $N = 5(1, 1, 2)$ and the formula $\varphi_3 \equiv \langle\langle c12 \rangle\rangle F\, \mathsf{castle3Damaged}$ which says that Workers 1 and 2 have a collective strategy to decrease the HP of castle 3. For presentation purposes we have shortened the representation of agents' local states, e.g., we write "FFF" instead of "Environment.castle1Defeated = false, Environment.castle2Defeated = false, Environment.castle3Defeated = false".

While performing verification with (unbounded) path-based strategy search, the following solution was found after 2 attempts:

```
Agent Worker1 - Generated strategy:
(FFF, Worker1.canDefend = true): {defend}
(FFF, Worker1.canDefend = false): {attack3}
(TFT, Worker1.canDefend = true): {doNothing}

Agent Worker2 - Generated strategy:
(FFF, Worker2.canDefend = true): {attack3}
(TFF, Worker2.canDefend = true): {defend}
(TFT, Worker2.canDefend = true): {doNothing}
```

We also performed verification with bounded path-based strategy search. The following solution was found after 12 attempts:

```
Agent Worker1 - Generated strategy:
(FFF, Worker1.canDefend = true): {attack3}
Agent Worker2 - Generated strategy:
(FFF, Worker2.canDefend = true): {attack3}
```

6 Conclusions

Verification of strategic abilities under imperfect information has been extensively studied theoretically, but at the same time ignored as far as practical algorithms and tools are concerned. This paper reports our first step towards filling the gap. We propose and implement an algorithm for model checking ATL_{ir}, i.e., the variant of alternating-time logic based on uniform positional strategies. The experimental results are encouraging. In particular, our algorithm significantly outperformed the only other existing tool (an experimental version of MCMAS), despite the fact that MCMAS uses symbolic model checking techniques based on OBDD's, and our SMC operates purely on explicit representations of states.

Our algorithm enables speedup coming from two potential sources. First, it considers only so called proper strategies which are in fact equivalence classes of concrete strategies. A variant of SMC restricts the search even further by considering only so called path-based strategies. Secondly, strategies are sought incrementally, starting from simplest ones. In many scenarios, whenever a successful strategy exists, it can be found among the relatively simple ones. In those cases, our algorithm finds a good strategy after a number of attempts vastly smaller than the number of all proper strategies in the model. In the experiments, the first kind of speedup yielded reductions of the search space by order of 10^6 times up to 10^{12} times. The second kind of speedup yielded solutions after no more than 10 attempts for problems where the number of proper strategies ranged from order of 10^2 to 10^5. As a result, the strategy verification sub-routine was called only around $10^0 = 1$ times, yielding a speedup of the verification stage of order of 10^2 up to 10^5.

Despite the promising experimental results, our tests showed also that the problem itself *is* computationally difficult. We observed an overwhelming gap in performance between verification of strategic abilities for perfect vs. imperfect information strategies. On the other hand, there is still much room for improvement. In particular, we plan to employ symbolic model checking techniques (based on OBDD's and/or translation to SAT solvers) as well as parallelization using e.g. the DACFrame, Akka, or GridGain platforms for parallel computation (cf. also [15]). Further future work includes extending the syntax accepted by SMC to all ATL_{ir} formulae in negation normal form, more experiments with various benchmark models and formulae, and an extensive case study on an example of practical interest, e.g., verification of privacy and noninterference in a voting protocol. For the last task, an appropriate abstraction will have to be developed, possibly along the lines of [14].

Acknowledgements. This contribution has been supported by the Foundation for Polish Science under International PhD Projects in Intelligent Computing; project financed from The European Union within the Innovative Economy Operational Programme 2007-2013 and European Regional Development Fund. Wojciech Jamroga acknowledges the support of the FNR (National Research Fund) Luxembourg under project GALOT – INTER/DFG/12/06. We gratefully acknowledge the help of Hongyang Qu and Alessio Lomuscio who kindly provided us with the latest experimental version of MCMAS, as well as MCMAS model generators.

References

1. T. Ågotnes. A note on syntactic characterization of incomplete information in ATEL. In *Procedings of Workshop on Knowledge and Games*, pages 34–42, 2004.
2. T. Ågotnes, V. Goranko, W. Jamroga, and M. Wooldridge. Knowledge and ability. In W. van der Hoek H.P. van Ditmarsch, J.Y. Halpern and B.P. Kooi, editors, *Handbook of Logics for Knowledge and Belief*. Springer, 2014. To appear.
3. R. Alur, L. de Alfaro, R. Grossu, T.A. Henzinger, M. Kang, C.M. Kirsch, R. Majumdar, F.Y.C. Mang, and B.-Y. Wang. jMocha: A model-checking tool that exploits design structure. In *Proceedings of ICSE*, pages 835–836, 2001.
4. 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.
5. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.
6. R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, and S. Tasiran. MOCHA user manual. In *Proceedings of CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525, 1998.
7. N. Bulling and W. Jamroga. Comparing variants of strategic ability. *Journal of Autonomous Agents and Multi-Agent Systems*, 28(3):474–518, 2014.
8. J. Calta, D. Shkatov, and B.-H. Schlingloff. Finding uniform strategies for multi-agent systems. In *Proceedings of CLIMA*, volume 6245 of *Lecture Notes in Computer Science*, pages 135–152. Springer, 2010.

9. C. Dima and F.L. Tiplea. Model-checking atl under imperfect information and perfect recall semantics is undecidable. *CoRR*, abs/1102.4225, 2011.
10. W. Jamroga and T. Ågotnes. Modular interpreted systems: A preliminary report. Technical Report IfI-06-15, Clausthal University of Technology, 2006.
11. W. Jamroga and J. Dix. Model checking ATL$_{ir}$ is indeed Δ_2^P-complete. In *Proceedings of EUMAS'06*, 2006.
12. W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 63(2–3):185–219, 2004.
13. M. Kacprzak and W. Penczek. Unbounded model checking for Alternating-time Temporal Logic. In *Proceedings of AAMAS-04*, 2004.
14. M. Köster and P. Lohmann. Abstraction for model checking modular interpreted systems over ATL. In *Proceedings of AAMAS*, pages 1129–1130, 2011.
15. M. Kwiatkowska, A. Lomuscio, and H. Qu. Parallel model checking for temporal epistemic logic. In *Proceedings of ECAI*, pages 543–548, 2010.
16. A. Lomuscio, H. Qu, and F. Raimondi. MCMAS : A model checker for the verification multi-agent systems. In *Proceedings of CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 682–âĂŞ688, 2009.
17. A. Lomuscio and F. Raimondi. MCMAS : A model checker for multi-agent systems. In *Proceedings of TACAS*, volume 4314 of *LNCS*, pages 450–454, 2006.
18. P. Papalamprou. Logic-based verification of games with imperfect information. Master thesis, University of Luxembourg, 2013.
19. H. Qu, A. Lomuscio, and F. Raimondi. MCMAS with uniform strategies. Personal communication, 2014.
20. P. Y. Schobbens. Alternating-time logic with imperfect recall. *Electronic Notes in Theoretical Computer Science*, 85(2):82–93, 2004.
21. W. van der Hoek, A. Lomuscio, and M. Wooldridge. On the complexity of practical ATL model checking. In P. Stone and G. Weiss, editors, *Proceedings of AAMAS'06*, pages 201–208, 2006.