SMC: Synthesis of Uniform Strategies and Verification of Strategic Ability for Multi-Agent Systems

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

[1] Systems Research Institute, Polish Academy of Sciences
[2] Polish-Japanese Academy of Information Technology
[3] Institute of Computer Science, Polish Academy of Sciences
[4] Computer Science and Communication, and Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg

Abstract. We present a model checking algorithm for a subset of 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 (**NP**- to **PSPACE**-complete, depending on the assumptions), but very few *practical* attempts at it have been proposed so far. Our algorithm searches through the set of possible uniform strategies, utilizing a simple technique that reduces the search space. In consequence, it does not only verify 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. We also discuss two variants of the model checking problem, related to the objective vs. subjective interpretation of strategic ability. We provide algorithms for reductions between the two semantic variants of model checking. The algorithms are experimentally validated as well.

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 of 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 [32, 15, 2, 18], meta-logical properties [9], and the complexity of model checking [32, 17, 16]. 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 [32, 17] and undecidable for agents with perfect recall of the past [13], see also [8] for an overview of complexity results. Moreover, the imperfect information semantics of ATL does not admit fixpoint equivalences [9], which makes incremental synthesis of strategies impossible, or at least cumbersome. Still, some other results [16, 33] 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 [7, 4, 20, 19, 24, 23]. Either way, we believe that the scientific approach requires an extensive study of the practical hardness of the problem. This work 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 $\langle\langle C\rangle\rangle\varphi$, the algorithm tries to synthesize an executable strategy[5] for agents in the set C, that would enforce property φ. 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 a modified version of MCMAS [1, 31]. 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 [4, 24, 23]. We compare the performance of our algorithm to one of those tools (the standard version of MCMAS [23]) 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 [12] has been studied in [28], with results that suggested very bad performance.

Furthermore we discuss two semantic variants of the model checking problem. The first one, based on the notion of *objective ability* [15, 9], requires a successful strategy to exist for the initial global state of the system. The other one, based on the *subjective* interpretation of ability [32, 18], requires the strategy to exist for a certain set of states (namely, the states that are indistinguishable from the objective initial state). We provide translations between the two variants of model checking of ATL_{ir}. Besides theoretical interest, the translations enable

[5] such strategies are often called *uniform*.

model checking of both semantic variants with algorithms designed for either variant.

Previous versions of the material. This article is based on the conference papers [29, 30]. In this version, we add more explanations and discussions of the ideas and results. We also present revised experimental results. Moreover, we propose completely novel semantic translations between the subjective and objective semantics of ATL_{ir}, together with new experimental results based on the translations.

Other related work. While this article was being prepared for the journal publication, some new attempts at practical model checking of ATL_{ir} have emerged [10, 11, 14]. Their experimental results seem comparable to ours. In particular, they confirm that verification of strategic modalities for imperfect information is hard, and the current algorithms and verification techniques are far from satisfactory. A more thorough comparison is outside of the scope of this paper.

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 [5, 6] for reasoning about abilities of agents in multi-agent systems.

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: $\Sigma = \{1, \ldots, k\}$ is a finite nonempty set of *players* (also called *agents*), Q is a finite nonempty set of *states*, Π is a finite set of atomic propositions, $\pi : Q \to 2^{\Pi}$ is the *labeling function*. Moreover, for each player $a \in \{1, \ldots, k\}$ 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 $1, \ldots, d_a(q)$. For each state $q \in Q$, a *move vector* at q is a tuple $\langle j_1, \ldots, j_k \rangle$ such that $1 \leq j_a \leq d_a(q)$ for each player a. Furthermore, $D(q)$ denotes the set $\{1, \ldots, d_1(q)\} \times \ldots \times \{1, \ldots, d_k(q)\}$ of move vectors. Finally, δ is the deterministic *transition function* that returns a state $q' = \delta(q, j_1, \ldots, j_k)$ for each $q \in Q$ and $\langle j_1, \ldots, j_k \rangle \in D(q)$.

Intuitively, formula $\langle\langle C \rangle\rangle \varphi$ expresses that the group of agents C has a collective strategy to enforce φ. The formal syntax of ATL is given by the following grammar:

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

where $p \in \Pi$ is an atomic proposition, $C \subseteq \Sigma$ is a set of agents, and the operators X, G, and $\mathcal{U}$ stand for "in the next state", "always from now on", and

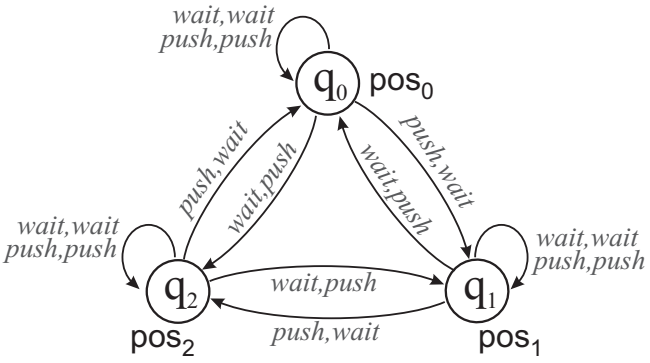

Fig. 1. Robots and carriage: concurrent game structure M_1

"strong until", respectively. Additional operator F ("eventually") can be defined as $F\varphi \equiv \top \mathcal{U} \varphi$.

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* S_C for agents $C \subseteq \Sigma$ is simply a tuple of strategies, one per agent in C.[7] Each collective strategy S_C induces a set of computations (paths, runs). Formally, by $out(q, S_C)$ we will denote the set of infinite sequences of states that can occur from state q on when the players in C follow strategy S_C and the other players are free to execute 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 $M, q \models \varphi_2$
- $M, q \models \langle\langle C \rangle\rangle X\varphi$ iff there exists a collective strategy S_C such that for all computations $\lambda \in out(q, S_C)$, we have $M, \lambda[1] \models \varphi$
- $M, q \models \langle\langle C \rangle\rangle G\varphi$ iff there exists a collective strategy S_C such that for all computations $\lambda \in out(q, S_C)$, and all positions $i \geq 0$, we have $M, \lambda[i] \models \varphi$
- $M, q \models \langle\langle C \rangle\rangle \varphi_1 U \varphi_2$ iff there exists S_C such that for all $\lambda \in out(q, S_C)$ 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 1. 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),

[6] We depart from the assumption in [5, 6] that agents have perfect recall of the past situations. Note that both types of strategies (memoryless and perfect recall) yield equivalent semantics in case of standard ATL [6, 32].

[7] In the rest of the paper, we will use S_C to refer to a collective strategy of coalition C, whereas s_a will be used to denote an individual strategy of agent a.

$\neg\langle\langle 1\rangle\rangle F\mathsf{pos}_1$ (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 global state of the system at each step of its execution is usually unrealistic; similarly, assuming perfect recall is not always practical [15, 32, 2, 18]. The differences between *perfect* and *imperfect information*, as well as between *perfect* and *imperfect recall*, gives rise to four different semantic variants of ATL, cf. [32, 3]. 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 **i**nformation + imperfect **r**ecall 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. Formally, we do it by replacing the semantic clauses for $\langle\langle C\rangle\rangle\varphi$ with the following ones:

- $M, q \models \langle\langle C\rangle\rangle X\varphi$ iff there exists a collective *uniform* strategy S_C such that for all computations $\lambda \in out(q, S_C)$, we have $M, \lambda[1] \models \varphi$
- $M, q \models \langle\langle C\rangle\rangle G\varphi$ iff there exists a collective *uniform* strategy S_C such that for all computations $\lambda \in out(q, S_C)$, and all positions $i \geq 0$, we have $M, \lambda[i] \models \varphi$
- $M, q \models \langle\langle C\rangle\rangle \varphi_1 U \varphi_2$ iff there exists *uniform* S_C such that for all $\lambda \in out(q, S_C)$ there is $i \geq 0$ with $M, \lambda[i] \models \varphi_2$ and for all $0 \leq j < i$ we have $M, \lambda[j] \models \varphi_1$.

Note that this is the so called "objective" semantics of ATL_{ir} [15, 9], see also the extensive discussion after the example. The "subjective" semantics of ATL_{ir} [32] will be presented formally in Section 6.

Example 2. An example iCGS is depicted in Figure 2. 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.

Objective vs. Subjective Ability. The community of artificial intelligence recognized the importance of the concept of ability already in the 1960s. Since the beginning, ability and information were seen as closely connected. However, there is no single semantics of ability under imperfect information. On the contrary, a multitude of different semantics have been proposed over the years, each of them with a different rationale and applicability. This began already with the

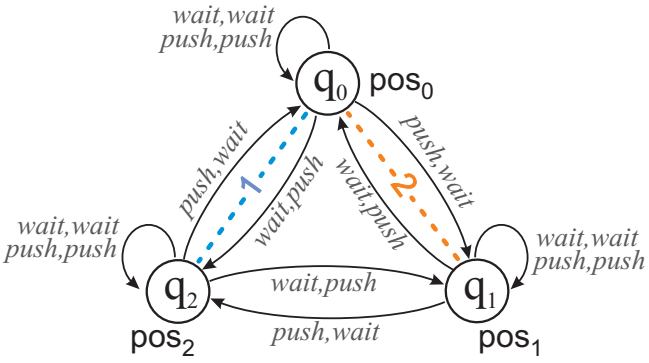

Fig. 2. Robots and carriage under uncertainty: iCGS M_2

seminal work of McCarthy and Hayes [25] where they proposed three different interpretations of what it means that an artificial agent θ (a *computer program* in their phrasing) is able to bring about the state of affairs φ:

1. There is a sub-program σ and room for it in memory which would achieve φ if it were in memory, and control were transferred to θ. No assertion is made that θ knows σ or even knows that σ exists.
2. σ exists as above and that σ will achieve φ follows from information in memory according to a proof that θ is capable of checking.
3. θ's standard problem-solving procedure will find σ if achieving φ is ever accepted as a subgoal.

Similar considerations shaped the other most influential treatment of ability in AI by Moore [26, 27].

The three cases address three viable interpretations of ability for autonomous agents. In case (1), ability is viewed from the standpoint of an omniscient external observer who can see that there is some procedure such that, if agent θ follows the procedure, then the achievement of φ will result. Thus, it corresponds to *objective* ability of the agent to bring about φ. Case (2) implies knowledge of the fact on the part of the agent (at least in the theoretical sense of modal epistemic logic). It is sometimes referred to as *subjective* ability. Interpretation (3) refers to *practical* ability: not only does the possibility for the agent to achieve φ exist, but the agent is capable of computing and executing an appropriate strategy σ. Thus, the last interpretation is algorithmic, and may be formalized through the computational problem of *strategy synthesis*.

In this paper, we begin with the *objective* semantics of ability, as formalized by the semantic clauses above, in order to simplify things computationally. That is, in order to evaluate the truth of $\langle\langle C\rangle\rangle\varphi$, we look only at the outcome paths starting from the current *objective* state of the system. Note that the semantics differs slightly from the "classical" *subjective* semantics of ATL$_{ir}$ proposed in [32]. On the other hand, both semantics coincide for models where the agents *know*

exactly the initial state of the system. We will come back to this issue more formally in Sections 6 and 7, where we will deal with the subjective semantics of ATL$_{ir}$. More precisely, we will propose translations between the objective and the subjective semantics, and show how they influence the performance of model checking.

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 [5, 6]. Model checking of ATL$_{ir}$ is much harder, namely $\mathbf{\Delta_2^P}$-complete [32, 17]. Moreover, for formulae with a single non-negated coalitional modality it becomes **NP**-complete [32]. The increase in complexity is mainly because fixpoint characterizations of strategic modalities do *not* hold under imperfect information [9], 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 [32] that model-checks formula $\langle\langle C\rangle\rangle\varphi$ in M, q:

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

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

In order to construct a deterministic version of the algorithm, we need a fixed 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.[8] Since the space of solutions is computationally large, it is crucial that we restrict the search space as much as possible. We will reduce the search space by identifying some equivalences between solutions.

Definition 1. *For a model M and strategy S for coalition C, we define a* trimmed model M_S *as a restriction of model M, where agents from coalition C 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.

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

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

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.

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

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 $q \in dom(s_a)$, we have either $s_a(q) = \{1, \ldots, d(q)\}$ or s_a is a singleton.*

The explicit part *of a partial strategy s_a is the restriction of s_a to $q \in Q$ where $s_a(q)$ is a singleton. The* implicit part *of a partial strategy s_a is the restriction of s_a to $q \in Q$ where $s_a(q) = \{1, \ldots, d(q)\}$. 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_a, denoted $|s_a|$, as the number of indistinguishability classes of states contained in $dom(s_a)$. A partial strategy s_a is* empty *iff $|expl(s_a)| = 0$. Conversely, s_a is* fully determined *iff $|impl(s_a)| = 0$.*

In a model M, the move function d_a determines the sets of actions available to agent a 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_a).

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

The empty strategy S_0 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 S_1 is defined in the following way: $S_1(x) = \{$push if $x =$ color, wait if $x =$ noColor$\}$. S_1 assigns a single action for all states in Q, leaving the implicit strategy empty (of size 0). An example partial strategy S_2 is defined in the explicit part as follows: $expl(S_2)(x) = \{push\}$ if $x = color$. Therefore the implicit part of the strategy is: $impl(S_2)(x) = \{push, wait\}$ 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 $C \subseteq \Sigma$ is simply a tuple of partial strategies for $a \in C$. It is empty iff all its components are empty, fully determined iff all its components are determined, etc.

3.3 Heuristics: 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 only at such strategies, 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* line *iff there is a transition in* M *between every* q_i *and* q_{i+1}.

We call a line $(q_1, \ldots, q_n)$ *a* lasso *in* M *iff 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.

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.

We will use the concepts to define simple heuristics for our model checking algorithm. In essence, the algorithm will implement three different variants of strategy search:

Branching search: search through all the proper strategies,
Path-based search: search only through path-based strategies,
Bounded path-based search: search only through bounded path-based strategies.

Note that the path-based search does not assume that only states on a lasso must be reachable by a successful strategy. Rather, it is about searching through strategies that only constrain choices along some lasso. States outside the lasso can be also reachable, but the strategy does not constrain agents' choices there, so the temporal property must in fact hold on all paths outside the lasso.

4 The SMC Model Checker

SMC (Strategic Model Checker)[9] is a software tool designed for model checking ATL_{ir} and synthesis of uniform strategies.

The current version of SMC can check ATL_{ir} formulae that contain at most one coalitional modality. More precisely, the following formulae classes are supported:

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

where φ, φ' are propositional Boolean formulae.

Remark 1. An extension to the full logic ATL_{ir} is planned for future work. We note, however, that the extension would require much more efficient verification techniques than are available now. In theory, model checking of formulae with nested strategic modalities is a straightforward extension of the simple case,

[9] SMC is available online at http://icr.uni.lu/wjamroga/smc.html

where model checking of subformulae is called recursively. Unfortunately, such an extension of our current algorithms would run into timeout for all but the most trivial model checking instances.

We also observe that specifications with nested modalities tend to occur less frequently than simple strategic properties. Consider, for instance, formula $\langle\langle C\rangle\rangle F\langle\langle B\rangle\rangle G\mathsf{p}$. It refers to C's ability to enable some ability of B – in this case, to maintain p forever. Such meta-abilities (i.e., ability to enable or disable ability) are important in some application domains (e.g., access control), but much more often one wants to make sure that some agents C can bring about or maintain a *factual* state of affairs p. That can be done by verifying the simple formula $\langle\langle C\rangle\rangle F\mathsf{p}$ (respectively, $\langle\langle C\rangle\rangle G\mathsf{p}$).

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. A pseudocode transcription of the algorithm can be found in the appendix.

4.1 High-Level Description of the Algorithm

The general structure of the algorithm is as follows:

1. For a formula of type $\langle\langle C\rangle\rangle\varphi$, synthesize a previously unverified strategy S_C to be verified;
2. Check the CTL formula $\mathsf{A}\varphi$ in the trimmed model M_{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 step 1.

Note that memoryless strategies are of polynomial size with respect to the number of global states and transitions in the model. Thus, the above algorithm runs in nondeterministic polynomial time.

Step 1 (strategy synthesis step) is the most significant, as step 2 can be performed by 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 self-explanatory. Thus, our next move is to elaborate on step 1:

1.1 Start with the empty partial strategy $S_C = \emptyset$ and with the initial state q_0;
1.2 In a loop, generate potential partial strategies by extending S_C. To do this, fix the actions of agents in C for a newly discovered state, reachable by S_C, that does not have fixed actions yet;
1.3 Continue step 1.2 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* $ST = \langle F, U, S_C\rangle$ consists of:

- The set of *fixed states* F. For any state in F, the actions of all agents from C are already fixed in the explicit domain of the partial strategy S_C;
- The set of *unchecked states* U. Each state in U has no explicit actions assigned in S_C for some or all of the agents in C;
- The partial strategy S_C.

By STL, we will denote a list of strategy tasks. We will implement STL as a sequential data structure (queue) that stores the strategy tasks to be processed. The list is initialized with $STL_0 = \{\langle \emptyset, \{q_0\}, S_\emptyset \rangle\}$, where q_0 is the initial state and $S_\emptyset$ is the empty partial strategy.

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. The current strategy task will be referred to as $CST = \langle F, U, S_C \rangle$;
2. Fix the current state $CS \in U$ and do $F := F \cup \{CS\}$, $U := U \setminus \{CS\}$;
3. Generate all possible children strategies for S_C, reachable by fixing an action for every agent in C at the current state CS, with the exception of agents that already have an action fixed for CS.
   Note: Some agents in the coalition may have an action fixed for the state CS (due to indistinguishability). The already fixed actions remain intact. Note also that this can apply to all agents in the coalition, therefore it is possible that this step generates no new strategies.
4. If there were no new strategies generated in step 3, generate a new strategy task $\langle F, U, S_C \rangle$. (Note: the strategy is still S_C, but we have changed F and U in step 2.) Add this strategy task to STL if $U \neq \emptyset$. Return to step 1.
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-based 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 branching (respectively, unbounded path-based) strategy search is enabled, ignore this step unless the current strategy is a branching (respectively, unbounded path-based) strategy. Otherwise, pass the current strategy to the verification step (done by means of CTL fixpoint model checking of the trimmed model M_{S_C}). If the verification returns *true*, terminate with answer *strategy found* and return the current strategy as the 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 the partial strategy. For any agent in the coalition, we add a single entry that fixes the action in the currently processed state, unless such an action is already fixed. The possibility of the 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 from a state that has been processed before. Then, the agent has an already fixed action for the whole equivalence class containing both states. Step 4 describes the special case where we have a previously unchecked global state that has already fixed actions for all agents in the coalition.

Step 5 describes a situation where at least one of the agents has no fixed action for the current state, and therefore has the possibility to extend his partial strategy by adding a new entry to the explicit domain of the strategy. If path-based synthesis is enabled, the algorithm only considers as sources of strategy refinement states reachable from the current state (CS). This is achieved by setting U to $\emptyset$. In consequence, only one successor of the current state CS is used to extend a strategy, and all the other successors are forgotten. This leads to path-based strategies.

4.3 Discussion

Our approach enables 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 $\langle\langle a \rangle\rangle F$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 sometimes it is not necessary to build fully determined and/or unbounded strategies.

A proper partial strategy can be described by the explicit strategy part. For instance, the output in Example 4 can be represented by a bounded path-based strategy of size 1. In contrast, generating a fully determined strategy could easily require a domain size of 10^3 or more. The more compact representation has two obvious advantages: reduced memory/processing requirements for computers and improved readability of the output for humans.

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.
– *SMC with bounded path-based strategy search* searches only through bounded path-based strategies.

We call a variant of SMC *sound* iff $SMC(M, q, \langle\langle C \rangle\rangle \varphi) = true$ implies $M, q, \langle\langle C \rangle\rangle \models \varphi$. Conversely, the variant of SMC is *complete* iff $M, q, \langle\langle C \rangle\rangle \models \varphi$ implies $SMC(M, q, \langle\langle C \rangle\rangle \varphi) = true$. 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.

The timeout was set to 120 minutes.

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 and regain its strength 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 coalition of agents working for castles 1 and 2 has 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.

N	Total time (s)	1st step (s)	2nd step (s)	Peak memory (MB)
4 (1 1 1)	0.1	0.1	<0.1	15
5 (2 1 1)	0.2	0.2	<0.1	26
6 (2 1 2)	4.5	0.5	4	606
7 (2 2 2)	3.4	2.6	0.7	77
8 (3 2 2)	255.5	27	228.5	454
10 (3 3 3)	timeout	timeout	timeout	timeout

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 instance size N: the total number of agents (including 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 seconds (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.[10]

5.3 Number of Generated Strategies

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

[10] 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 (2 1 1)	3	$8.9 * 10^{12}$	16 616	1
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
10 (3 3 3)	6	$7.9 * 10^{25}$	not calculated	unknown

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

1. The instance size N;
2. The number of agents in the coalition for which a strategy is constructed;
3. The total number of strategies, as defined by the semantics of ATL_{ir};
4. The total number of *proper unbounded path-based* strategies;
5. The number of strategies processed by the algorithm.

5.4 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 seconds) for formula $\varphi_1 \equiv \langle\langle c12 \rangle\rangle F\mathsf{castle3Defeated}$ in different instances of the class of models.

N	Path-based strategy search	Branching strategy search
4 (1 1 1)	0.1 s	0.8 s
5 (2 1 1)	0.2 s	3.7 s
6 (2 1 2)	4.5 s	72.4 s
7 (2 2 2)	3.4 s	261.7 s
8 (3 2 2)	255.6 s	2 122 s
10 (3 3 3)	timeout	timeout

Thus, in the above experiments, finding an existing strategy was faster by an order of magnitude when using the path-based heuristics. The difference is much more dramatic when verifying a goal for which no successful strategy exists. Consider formula

$$\varphi_2 \equiv \langle\langle w12 \rangle\rangle F\mathsf{allDefeated}$$

which 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 formula does not hold in any of the tested models. In consequence, the complete variant of the algorithm must generate and check a huge number of all the proper strategies, whereas the incomplete variant terminates after checking the path-based strategies only:

N	Path-based strategy search	Branching strategy search
4 (1 1 1)	5.4 s	timeout
5 (2 1 1)	51 s	timeout

5.5 Comparison to MCMAS

The only tool for ATL$_{ir}$ model checking, that was available during our writing of the paper, has been an experimental version of MCMAS, kindly provided to us by its authors [1, 31]. A tentative comparison of SMC and MCMAS is presented below. All the parameters of the experiments are as in the preceding sections. We have used the following two formulae:

$\varphi_1 \equiv \langle\langle c12 \rangle\rangle F \mathsf{castle3Defeated}$ (same as before; *true* in the tested models)
$\varphi_2 \equiv \langle\langle w12 \rangle\rangle F \mathsf{allDefeated}$ (same as before; *false* in the tested models)

The tables below compare the performance of both model checkers where SMC uses the variant with path-based strategy search. As before, the timeout was set to 120 minutes.

N	Formula	MCMAS execution time	SMC (path-based) execution time	SMC (branching) execution time
4 (1 1 1)	φ_1	72 s	0.1 s	0.8 s
5 (2 1 1)	φ_1	timeout	0.2 s	3.7 s
4 (1 1 1)	φ_2	78 s	5.4 s	timeout
5 (2 1 1)	φ_2	error	51 s	timeout

N	Formula	MCMAS tested strats.	SMC tested strats. (path-based)
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. 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 $N = 5(2\ 1\ 1)$. We will come back to this issue in Section 7.

5.6 Perfect vs. Imperfect Information Strategies

One of the goals in this work was to determine how model checking of abilities under imperfect information compares in practice to the standard ATL case. To this end, we have compared the performance of SMC and the modified version of MCMAS to the default perfect information version of MCMAS [23]. The table reports model checking times (in seconds) for formula φ_1 in various instances of

the *Castles* class. As expected, verification in the perfect information case scales up much better, though it also displays exponential dynamics of the execution time.

N	perfect info (MCMAS)	imperfect info (SMC)	imperfect info (MCMAS)
4 (1 1 1)	<0.1 s	0.1 s	72 s
5 (2 1 1)	<0.1 s	0.2 s	timeout
6 (2 1 2)	0.3 s	4.5 s	timeout
7 (2 2 2)	1 s	3.4 s	timeout
8 (3 2 2)	2 s	255.6 s	timeout
10 (3 3 3)	20.4 s	timeout	timeout

5.7 Example Output of Strategy Synthesis

An important feature 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 From Objective to Subjective Semantics and Back

As we already mentioned in Section 2.2, there are at least two meaningful interpretations of ability under imperfect information, even if we assume that agents

use uniform memoryless strategies. On one hand, an agent is *objectively* able to enforce φ if he has an executable strategy that infallibly succeeds from the *real* initial state of the game. Note that this does not imply that the agent can come up with the right strategy. A stronger property requires that the agent can synthesize such a strategy on his own. In other words, there must be a strategy that enforces φ not only from the actual state of the system, but also from all the states that look the same to the agent. We refer the reader to [9] for a more detailed discussion.

Formally, the two semantics of ability can be defined as follows:

Objective semantics $M, q_0 \models_O \langle\langle C\rangle\rangle\varphi$ iff there exists a collective uniform strategy S_C for C such that for all computations $\lambda \in out(q_0, S_C)$, we have $M, \lambda \models \varphi$.

Subjective semantics $M, q_0 \models_S \langle\langle C\rangle\rangle\varphi$ iff there exists a collective uniform strategy S_C for C such that for all states $I = \{q \mid \exists_{a\in C}\ q \sim_a q_0\}$ for all $q \in I$ and all computations $\lambda \in out(q, S_C)$, we have $M, \lambda \models \varphi$.

The semantic relations $\models_O, \models_S$ for path formulae are defined like in CTL, e.g., $M, \lambda \models X\psi$ iff $M, \lambda[1] \models \psi$, etc.

Both semantics have their uses, and both are rooted in the philosophical discourse on strategic ability [25, 26, 3]. Choosing one or the other interpretation depends on the actual scenario, and most of all on the requirement that is to be verified. Thus, it is natural to seek tool support for model checking in both cases. The SMC model checker, proposed in this paper, handles the objective semantics. MCMAS [24, 23] does model checking of ATL_{ir} according to the subjective semantics. In this section, we propose two simple transformations of models that allow to "swap" the tools, i.e., to use MCMAS for model checking of objective ability, and SMC for verification of subjective ability. Among other things, this will help us to better compare the performance of both tools.

In the rest of this section we only consider formulae of ATL with one coalitional modality as the top level operator. More precisely, only formulae of the form $\langle\langle C\rangle\rangle Fp$, $\langle\langle C\rangle\rangle Gp$, $\langle\langle C\rangle\rangle p\,U\,r$, $\langle\langle C\rangle\rangle Xp$ are considered, where p and r are literals (i.e., atomic propositions or their negations).

6.1 Translations: Preliminaries

We describe truth-preserving translations between the subjective and objective semantics of ATL_{ir}. Both translations only transform the model, and leave the formula intact. The transformations follow the idea that the model checking of $\langle\langle C\rangle\rangle\varphi$ in (M, q_0) according to one semantics can be reduced to model checking of $\langle\langle C\rangle\rangle\varphi$ in (M', q_0') according to the other semantics. Formally, we will define model transformations $subj2obj$ and $obj2subj$, such that:

$$M, q_0 \models_S \langle\langle C\rangle\rangle\varphi \quad \text{iff} \quad M', q_0' \models_O \langle\langle C\rangle\rangle\varphi, \text{ where } (M', q_0') = subj2obj(M, q_0);$$

$$M, q_0 \models_O \langle\langle C\rangle\rangle\varphi \quad \text{iff} \quad M', q_0' \models_S \langle\langle C\rangle\rangle\varphi, \text{ where } (M', q_0') = obj2subj(M, q_0).$$

For both *subj*2*obj* and *obj*2*subj*, we construct the new model M' in a relatively simple way. We start by augmenting the old model M with a fresh state q_0' which will serve as the new initial state. Furthermore, we extend the set of agents in M by adding a fresh agent Dec (the *"decider"*).

Let q be a state in the old model M. The actions available in M' at q to all agents other than Dec are the same as in M. The new agent Dec has only one action ϖ available (the trivial or "no operation" action). The transition function in M' at q cuts down to the transition function in M at q, i.e., $\delta'(q,(m,\varpi)) = \delta(q,m)$. Interpretation of atomic propositions at q in M' is the same as in M.

It remains now to define the structure of M' at the new state q_0'. That is, we must: (i) explain what actions are available in M' at q_0' to all agents, (ii) define the transitions that can occur in q_0', and (iii) provide the interpretations of atomic propositions at q_0'. In general, this will depend on the direction of the translation *and* on the temporal operator in formula φ. We present the detailed constructions in Sections 6.2 and 6.3. At this point, we only emphasize that none of the constructions introduces a transition from an "old" state q to the new state q_0'. Thus, q_0' is not reachable from any other state in M', and in consequence any computation in M' can contain q_0' at most once.

6.2 From Subjective Semantics to Objective Semantics (*subj*2*obj*)

In this subsection, we describe how to construct $(M', q_0') = subj2obj(M, q_0)$. Let $I = \{q \in M \mid \exists_{a \in C}\ q \sim_a q_0\}$ denote the set of states of M which are indistinguishable from q_0 to at least one agent in C. The construction depends on the temporal operator in the model-checked formula $\langle\langle C\rangle\rangle\varphi$.

Temporal operators F, G, U. We add the new state q_0' in such a way that the decider has total control over the first transition of the system. Formally, all agents other than Dec can execute only action ϖ.Moreover, $d_{Dec}(q_0') = I$. Finally, $\delta'(q_0', (\varpi, \ldots, \varpi, q)) = q$. That is, the decider points out one of the states in I, and the system proceeds accordingly.

In consequence, all and only the states in I are reachable from $q_0{}'$ in the first step of any computation of M'.

The evaluation of atomic propositions at q_0' is defined below in such a way that for any computation λ from q_0 in M, prefixing λ with $q_0{}'$ yields a valid computation in M' that does not affect the satisfiability of the model checked formula. Formally, for atomic propositions $p, r \in \Pi$ do the following.

- Make p false at $q_0{}'$, for formulae of type $\langle\langle C\rangle\rangle Fp$;
- Make p true at $q_0{}'$, for formulae of type $\langle\langle C\rangle\rangle Gp$;
- Make p true and r false at $q_0{}'$ for formula of type $\langle\langle C\rangle\rangle p\,Ur$.[11]

[11] For formulae $\langle\langle C\rangle\rangle p\,Up$, we make p true at $q_0{}'$ if $\forall_{q \in I} M, q \models p$, and false otherwise.

Temporal operator X. Let $\mathcal{S}_a$ be the set of *uniform strategies of agent a on states I*. Formally,

$$\mathcal{S}_a = \{S : I \to \mathbb{N} \mid \forall_{q \in I}\, S(q) \in d_a(q) \wedge \forall_{q,r \in I}\, q \sim_a r \Rightarrow S(q) = S(r)\}.$$

$\mathcal{S}_a$ will play the role of the set of actions available to a at q_0', for all $a \in C$. The other agents from M have only the "no operation" action ϖ available at q_0'. Moreover, the actions of Dec at the new state are constructed as tuples $(q, \alpha_{a_1}, \ldots, \alpha_{a_n})$ where:

- $q \in I$,
- $\overline{C} = \{a_1, \ldots, a_n\}$ are the agents from M that are not in C,
- For each of them, $\alpha_{a_i} \in d_M(q, a_i)$.

Finally, take any move vector m' such that $m'_a = S_a$ for $a \in C$, $m'_a = \varpi$ for $a \in \overline{C}$, and $m_{Dec} = (q, \alpha_{a_1}, \ldots, \alpha_{a_n})$. Then, $\delta'(q_0', m') = \delta(q, m)$ where $m_a = S_a(q)$ for $a \in C$ and $m_a = \alpha_a$ for $a \in \overline{C}$. That is, the agents in C choose their one-step strategies for the states in I, while the decider selects the initial state from I and fixes the actions of the opponents – and the system proceeds accordingly. The interpretation of atomic propositions in the new state q_0' is irrelevant.

In each of the above cases the following holds.

Theorem 3. $M, q_0 \models_S \langle\langle C \rangle\rangle \varphi \quad \textit{iff} \quad M', q_0' \models_O \langle\langle C \rangle\rangle \varphi$.

Proof. Straightforward by construction.

6.3 From Objective Semantics to Subjective Semantics (*obj2subj*)

In this subsection, we describe how to construct $(M', q_0') = obj2subj(M, q_0)$, which is even simpler.

Temporal operators F, G, U. All agents have only one action ϖ available at the new state q_0'. The resulting unique transition leads to q_0. The evaluation of atomic propositions at q_0' is defined as in Section 6.2. Again, for any computation λ from q_0 in M, prefixing λ with $q_0{}'$ yields a computation in M' that does not affect the truth of the model-checked formula $\langle\langle C \rangle\rangle \varphi$.

Temporal operator X. This time, let Q_C denote the set of all states reachable from q_0 in M. All agents other than Dec have only the action ϖ available at q_0' in M'. Moreover, the set of actions available to Dec at q_0' is equal to Q_C. Finally, let $\delta'(q_0, m) = q$ iff $m_{Dec} = q$. That is, the decider points out the next state, and the system proceeds accordingly. Again, evaluation of atomic propositions at q_0' is irrelevant.

Theorem 4. $M, q_0 \models_O \langle\langle C \rangle\rangle \varphi \quad \textit{iff} \quad M', q_0' \models_S \langle\langle C \rangle\rangle \varphi$.

Proof. Straightforward by construction.

7 Using the Translations: Experiments

Recall that SMC handles model checking of abilities according to the objective semantics, while MCMAS does the same for the subjective semantics of ATL_{ir}. Intuitively, one can use the translations from Section 6 to "swap" the tools, i.e., to verify objective abilities with MCMAS and subjective abilities with SMC. We will now present experimental results that show how both tools perform when coupled with the appropriate translation. Throughout this section, we use the same notation, the same class of models, and the same benchmark formulae as in Section 5. The timeout was set to 120 minutes.

Remark 2. Note that in both cases, i.e., for the objective as well as subjective semantics, the decision problem that we consider is *local model checking*. That is, the task is to determine the truth of formula φ in a given state q of the model. However, in the case of the subjective semantics, this requires to check existence of a strategy that succeeds not only from q, but also from all the states in the appropriate epistemic neighborhood of q.

7.1 Model Checking Objective Abilities by Translation to Subjective Semantics

First, we employed the translation function $obj2subj$ so that MCMAS can be used to model-check objective abilities of agents. For SMC, we used the unaltered model M since SMC uses objective semantics natively. For MCMAS, the model checking was performed on model $M' = obj2subj(M)$. The performance results for SMC include the variant with heuristics (path-based strategy search) as well as the sound and complete variant of the algorithm (branching strategy search).

N	Formula	MCMAS exec. time	SMC (path-based)	SMC (branching)
4 (1 1 1)	φ_1	1338 s	0.1 s	0.8 s
5 (2 1 1)	φ_1	timeout	0.2 s	3.7 s
6 (2 1 2)	φ_1	timeout	4.5 s	72.4 s

7.2 Model Checking Subjective Abilities by Translation to Objective Semantics

Next, we employed the translation function $subj2obj$ so that SMC can be used to model-check subjective abilities of agents. For MCMAS, we used the unaltered model M, since MCMAS uses subjective semantics natively. For SMC, the model checking was performed on model $M' = subj2obj(M)$.

N	Formula	MCMAS execution time	SMC execution time
4 (1 1 1)	φ_1	72 s	timeout
5 (2 1 1)	φ_1	timeout	timeout

Unfortunately, SMC was not even able to complete computation for the simplest of the models within the timeout of 120 minutes. Note that SMC used the sound and complete variant of the model checking algorithm (branching strategy search). We also ran the variant with heuristics, but unsurprisingly it found no strategy, as the path-based search is not compatible with massively branching models.

7.3 Summary

Our results indicate that model checking with the subjective semantics can be much harder than model checking objective abilities. For the class of models used in this study (*Castles*), SMC performed quite well as far as the objective semantics is considered. For the subjective semantics, however, the results were bad. On one hand, the path-based heuristics proved useless. On the other hand, the complete variant of the algorithm proved inefficient – either because the algorithm was not good enough, or because the translation $subj2obj$ does not suit SMC well.

8 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 were 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 generally outperformed the only other existing tool (a modified 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. We must note, however, that the version of SMC for model checking subjective abilities performed very badly – definitely worse than MCMAS.

Our algorithm enables speedup due to 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.

SMC generates strategies of bounded explicit domain size $k = 0, 1, 2, \ldots$. We note that this is somewhat similar to bounded model checking (BMC). Most often the explicit strategy size of a successful strategy is smaller than the entire domain of the model for this strategy. Furthermore, when a solution exists it can often be found *quickly*. The number of examined proper strategies before finding a successful one is often significantly smaller than the number of all considered proper strategies. One important difference to BMC is that our algorithm is complete with respect to the class of strategies being searched, i.e., it searches all proper strategies, be it bounded, unbounded or branching.

This article reports only the first step towards practical model checking of strategic abilities under imperfect information. There is 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 [22]). 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 [21].

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 National Research Fund (FNR) Luxembourg under the project GALOT (INTER/DFG/12/06), the support of the 7th Framework Programme of the European Union under the Marie Curie IEF project ReVINK (PIEF-GA-2012-626398), and the support of the National Centre for Research and Development (NCBR), Poland, under the PolLux project VoteVerif (POLLUX-IV/1/2016).

Last but not least, we gratefully acknowledge the help of Hongyang Qu and Alessio Lomuscio who provided us with the latest experimental version of MCMAS, as well as with MCMAS model generators.

References

1. MCMAS. Available online at http://vas.doc.ic.ac.uk/software/mcmas/.
2. T. Ågotnes. A note on syntactic characterization of incomplete information in ATEL. In *Procedings of Workshop on Knowledge and Games*, pages 34–42, 2004.
3. T. Ågotnes, V. Goranko, W. Jamroga, and M. Wooldridge. Knowledge and ability. In H.P. van Ditmarsch, J.Y. Halpern, W. van der Hoek, and B.P. Kooi, editors, *Handbook of Epistemic Logic*, pages 543–589. College Publications, 2015.

4. 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. IEEE Computer Society Press, 2001.
5. 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.
6. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.
7. 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.
8. N. Bulling, J. Dix, and W. Jamroga. Model checking logics of strategic ability: Complexity. In M. Dastani, K. Hindriks, and J.-J. Meyer, editors, *Specification and Verification of Multi-Agent Systems*, pages 125–159. Springer, 2010.
9. 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.
10. S. Busard, C. Pecheur, H. Qu, and F. Raimondi. Improving the model checking of strategies under partial observability and fairness constraints. In *Formal Methods and Software Engineering*, volume 8829 of *Lecture Notes in Computer Science*, pages 27–42. Springer, 2014.
11. S. Busard, C. Pecheur, H. Qu, and F. Raimondi. Reasoning about memoryless strategies under partial observability and unconditional fairness constraints. *Inf. Comput.*, 242:128–156, 2015.
12. 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.
13. C. Dima and F.L. Tiplea. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR*, abs/1102.4225, 2011.
14. X. Huang and R. van der Meyden. Symbolic model checking epistemic strategy logic. In *Proceedings of AAAI*, pages 1426–1432, 2014.
15. W. Jamroga. Some remarks on alternating temporal epistemic logic. In B. Dunin-Keplicz and R. Verbrugge, editors, *Proceedings of Formal Approaches to Multi-Agent Systems (FAMAS 2003)*, pages 133–140, 2003.
16. W. Jamroga and T. Ågotnes. Modular interpreted systems: A preliminary report. Technical Report IfI-06-15, Clausthal University of Technology, 2006.
17. W. Jamroga and J. Dix. Model checking ATL_{ir} is indeed Δ_2^P-complete. In *Proceedings of EUMAS'06*, volume 223 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.
18. W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 63(2–3):185–219, 2004.
19. M. Kacprzak, A. Lomuscio, and W. Penczek. Verification of multiagent systems via unbounded model checking. In *Proceedings of (AAMAS*.
20. M. Kacprzak and W. Penczek. Unbounded model checking for alternating-time temporal logic. In *Proceedings of AAMAS-04*, pages 646–653. IEEE Computer Society, 2004.
21. M. Köster and P. Lohmann. Abstraction for model checking modular interpreted systems over ATL. In *Proceedings of AAMAS*, pages 1129–1130. IFAAMAS, 2011.
22. M. Kwiatkowska, A. Lomuscio, and H. Qu. Parallel model checking for temporal epistemic logic. In *Proceedings of ECAI*, pages 543–548, 2010.

23. 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 682–688. Springer, 2009.
24. A. Lomuscio and F. Raimondi. MCMAS : A model checker for multi-agent systems. In *Proceedings of TACAS*, volume 4314 of *Lecture Notes in Computer Science*, pages 450–454. Springer, 2006.
25. J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.
26. R. C. Moore. Reasoning about knowledge and action. In *Proceedings IJCAI*, pages 223–227. William Kaufmann, 1977.
27. R.C. Moore. A formal theory of knowledge and action. In J. Hobbs and R.C. Moore, editors, *Formal Theories of the Commonsense World*. Ablex Publishing Corp., 1985.
28. P. Papalamprou. Logic-based verification of games with imperfect information. Master thesis, University of Luxembourg, 2013.
29. J. Pilecki, M.A. Bednarczyk, and W. Jamroga. Model checking properties of multi-agent systems with imperfect information and imperfect recall. In *Proceedings of IS*, volume 322 of *Advances in Intelligent Systems and Computing*, pages 415–426. Springer, 2014.
30. J. Pilecki, M.A. Bednarczyk, and W. Jamroga. Synthesis and verification of uniform strategies for multi-agent systems. In *Proceedings of CLIMA XV*, volume 8624 of *Lecture Notes in Computer Science*, pages 166–182. Springer, 2014.
31. H. Qu, A. Lomuscio, and F. Raimondi. MCMAS with uniform strategies. Personal communication, 2014.
32. P. Y. Schobbens. Alternating-time logic with imperfect recall. *Electronic Notes in Theoretical Computer Science*, 85(2):82–93, 2004.
33. W. van der Hoek, A. Lomuscio, and M. Wooldridge. On the complexity of practical ATL model checking. In *Proceedings of AAMAS'06*, pages 201–208. ACM, 2006.

Appendix: Pseudocode Transcription of SMC

The main algorithm:

Let q_0 be the initial state.
Let $C = \{1, \ldots, n\}$ be the checked coalition.
Let $S_\emptyset$ be the empty partial strategy.
Let a *strategy task* be a data structure with the following 3 elements:

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_C;
2. The set of *unchecked* states U. Each state in U has no explicit actions assigned in S_C yet for some or all of the agents;
3. The partial strategy S_C.

Let STL be a initially empty queue of *strategy tasks*.

STL.add$(< \emptyset, \{q_0\}, S_\emptyset >)$;
while $STL \neq \emptyset$ **do**
  $CST := STL[0]$; // $CST :=< F, U, S_C >$ ▷ 1.

```
    F := CST.F;
    U := CST.U;
    S_C := CST.S_C;
    STL.remove(0);
    CS := U[0];                                                    ▷ 2.
    F := F ∪ {CS};
    U := U \ {CS};
    newStrategies = childrenStrategies(CS, S_C);                   ▷ 3.
    if newStrategies = ∅ then                                      ▷ 4.
        if not U = ∅ then
            STL.add(new <F, U ∪ (next(CS, S_C) \ F), S_C>);
        end if
    else                                                           ▷ 5.
        S_C := newStrategies[0];
        for all newS ∈ newStrategies \ {S_C} do
            newU := ∅;
            if not onlyPathBasedStrategies then
                newU := newU ∪ U;
            end if
            STL.add(new < F, newU ∪ (next(CS, S_C) \ F), newS >);
        end for
    end if

    checkCurrent = true;                                           ▷ 6.
    if newStrategies = ∅ then
        checkCurrent = false;
    end if
    if checkUboundedStrategies and not isUnbounded(S_C) then
        checkCurrent = false;
    end if
    if checkCompleteStrategies and not isComplete(S_C) then
        checkCurrent = false;
    end if
    if checkCurrent then
        if CTLCheck(S_C) then
            return STRATEGYFOUND(S_C);
        end if
    end if
end while
return NOSTRATEGYFOUND;
```

Auxiliary functions:

```
function CHILDRENSTRATEGIES(Strategy S_C, State CS)
    Let n := |C|;
```

Let $possibleActions$:= array[1..n];
Let $children := \emptyset$;
for all $a \in C$ **do**
if $S_C([CS]_{\sim_a})$ is defined explicitly **then**
$possibleActions[a] := S_a([CS]_{\sim_a})$;
else
$possibleActions[a] := \{1, \ldots, d_a([CS]_{\sim_a})\}$;
end if
end for
for all $actions \in possibleActions[1] \times \ldots \times possibleActions[n]$ **do**
$newS$:= copyOf(S_C); //S_C needs to be immutable.
for all $a \in C$ **do**
$newS_a([CS]_{\sim_a}) \leftarrow actions[a]$;
end for
$children = children \cup \{newS\}$;
end for
return $children \setminus \{S_C\}$;
end function

function NEXT(Strategy S_C, State CS)
Let $next = \emptyset$;
Let $allActions = \{1, \ldots, d_1([CS]_{\sim_1})\} \times \ldots \times \{1, \ldots, d_k([CS]_{\sim_k})\}$;
for all $\langle action_1, \ldots, action_k \rangle \in allActions$ **do**
if $\langle action_1, \ldots, action_k \rangle$ not *conflicting* $S_C(CS)$ **then**
(where *conflicting* $\equiv$ assigning an action other than $S_a([CS]_{\sim_a})$ to agent $a \in C$)
$next := next \cup \{\delta(CS, action_1, \ldots, action_k)\}$
end if
end for
return $next$
end function