ABSTRACT
We present an experimental tool for verification of strategic abilities under imperfect information, as well as strategy synthesis. The problem is well known to be hard, both theoretically and in practice. The tool, called Strategic Verifier (STV), implements several recently developed algorithms to overcome the complexity.

KEYWORDS
formal methods, alternating-time temporal logic, imperfect information, strategy synthesis, model checking

1 INTRODUCTION
As the systems around us become more complex, and at the same time more autonomous, the need for unambiguous specification and automated verification rapidly increases. Logics of strategic reasoning provide powerful tools to reason about various aspects of MAS [1, 3, 26, 31]. A typical property that can be expressed says that the group of agents A has a collective strategy to enforce temporal property ϕ, no matter what the other agents in the system do. Specifications in agent logics can be then used as input to model checking, which makes it possible to verify the correct behavior of a multi-agent system by an automated tool [9, 10, 13, 23].

Verification of strategic abilities is difficult for a number of reasons. The prohibitive complexity of model checking and strategy synthesis is a well known factor [5, 12, 25], which can be alleviated only to some degree by using symbolic data structures [4, 7, 13, 28]. Things become even harder for agents with imperfect information. The complexity ranges from NP-complete to undecidable [14, 31].

Even more importantly, fixpoint equivalences do not hold [6, 11], which makes the application of standard fixpoint algorithms invalid and the use of symbolic methods questionable. Most known approaches boil down to iteration over all the possible strategies [8, 24, 27]. Unfortunately, the number of available strategies is enormous.

Our team at PAS has recently developed two novel techniques that try to overcome the complexity [15, 17, 21]. In this short paper, we present an experimental tool STV that implements the techniques, together with a number of verification scenarios. The implementation is still preliminary (e.g., it does not provide a flexible input specification language). Still, it already allows to "play" with the verification problem, test the scalability of the new techniques, and visualize the complexity of models and strategies on intuitive benchmark scenarios.

2 APPLICATION DOMAIN
STV is aimed at verification of strategic abilities in multi-agent systems, and synthesis of strategies that guarantee a given temporal goal. Many relevant properties of MAS refer to abilities of agents and their groups. In particular, most functionality requirements can be specified as the ability of the authorized users to achieve their goals. At the same time, many security properties can be phrased in terms of the inability of unauthorized users to compromise the system. Concrete examples include:

- Formalizations of individual and group responsibility [33, 34],
- Functionality properties for teams of logistic robots, operating in an industrial environment [22, 30],
- Properties of receipt-freeness, coercion-resistance and voter-verifiability in voting procedures [2, 16, 32],
- Fairness in contract-signing protocols and non-repudiation protocols [19, 20],
- Existence of winning strategies in general games [29], as well as specific multi-player games such as Bridge [15, 17].

3 SCENARIOS
The tool includes the following verification scenarios:

1. Existence of a winning strategy in the ancient story of TianJi [23].
2. Ability of a team of "workers" to defeat a given castle in the Castles benchmark from [27].
3. Existence of a winning strategy for the declarer in the card game of Bridge (Bridge Endplay [15]).
4. Ability of a team of drones to visit a given number of locations (the Drones benchmark [18]).
5. A variant of coercion-resistance in a simple voting protocol (Simple Voting [15]).

4 FORMAL BACKGROUND
Models. The main part of the input is given by an imperfect information concurrent game structure [1, 31], i.e., a labeled multi-agent transition system with the transitions labeled by synchronous actions from all the agents in the system. The knowledge of each agent is represented by its epistemic indistinguishability relation. An example model is shown in Figure 1.
5 TECHNOLOGY

STV does explicit-state model checking. That is, the states and transitions of the model are represented explicitly in the memory of the tool. We have implemented model generators for the scenarios presented in Section 3: the user sets the values of the scaling parameters (e.g., the number of drones and their initial level of energy), and the corresponding model is generated. After that, two approaches to model checking can be selected: fixpoint approximation and dominance-based strategy search.

Approximate fixpoint verification [15, 17]. The first approach is based on computing fixpoint approximations of the verified formula. Two formulas are produced for $\langle A \rangle y$:

- The lower approximation $tr_L(\langle A \rangle y)$ is a fixpoint expression in an extension of alternating epistemic $\mu$-calculus [6] such that, if $tr_L(\langle A \rangle y)$ holds, then $\langle A \rangle y$ must hold as well.
- The upper approximation $tr_U(\langle A \rangle y)$ asks for perfect information strategies instead of imperfect information ones in the semantics of $\langle A \rangle$. Thus, whenever $tr_U(\langle A \rangle y)$ returns false, $\langle A \rangle y$ must be false, too.

Depth-first search with removal of dominated strategies [21]. The second technique is based on a novel notion of strategic dominance, applied in an incremental, DFS-based search for a winning strategy. We say that partial strategy $A_1$ dominates $A_2$ w.r.t. the context $A^g$ iff: (i) $A_1$ and $A_2$ share the same set of input states, and (ii) for each input state $q$, the set of possible output states of $A_1$ is a subset of those for $A_2$. In other words, $A_1$ is “tighter” than $A_2$, and induces a smaller set of outcome paths.

The algorithm, called DominoDFS, attempts to expand the context strategy that contains the initial state by exploring its frontier. For each state at the current frontier, DominoDFS collects all the available one-step strategies, and then removes the dominated ones. Besides the basic version, we have implemented several heuristics that determine the order of the search.

Implementation and evaluation. STV is implemented in Python 3. The algorithms have been evaluated on several benchmarks, with very promising results [15, 17, 21]. We used the state of the art model checker MCMAS [23] and the experimental tool SMC [27] as reference points. The results for Bridge endplay and Castles are shown in Tables 1 and 2, with the irrelevant columns omitted from the tables (fixpoint approximation is not applicable to Castles, and Bridge Endplay cannot be correctly encoded in SMC).

6 USAGE

The current version of STV (which can be found here) allows to:

- Select a class of predefined parameterised models and a predefined formula for verification (cf. Section 3).
- Set the values of the parameters that control scalability,
- Generate and display the explicit state-transition graph,
- Run the fixpoint approximation algorithm (lower and upper approximation),
- Run the dominance-based verification (DominoDFS),
- Display the verification result (truth value of the formula in the initial state of the model, states in the model where the formula holds, and possibly also the winning strategy that has been found).

7 CONCLUSIONS

Model checking strategic abilities under imperfect information is notoriously hard. Currently, no tools exist that would handle even toy examples in a satisfactory way. STV is our first step towards practical verification of such properties. We believe it is worth sharing with the MAS community even in this preliminary form.

Acknowledgements. The authors acknowledge the support of the National Centre for Research and Development (NCBR), Poland, under the PolLux project VoteVerif (POLUX-IV/1/2016).
REFERENCES


