SAT-Based ATL Satisfiability Checking

Magdalena Kacprzak[1], **Artur Niewiadomski**[2], **Wojciech Penczek**[3]
[1]Bialystok University of Technology, Bialystok, Poland
[2]Siedlce University, Faculty of Exact and Natural Sciences, Siedlce, Poland
[3]Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
m.kacprzak@pb.edu.pl, artur.niewiadomski@uph.edu.pl, penczek@ipipan.waw.pl

Abstract

Synthesis of models and strategies is an important task in software engineering. The main problem here consists in checking the satisfiability of formulae expressing the specification of a system to be implemented. This paper puts forward a novel method for deciding the satisfiability of formulae of Alternating-time Temporal Logic (ATL) under perfect and imperfect information. The method expands the one for CTL exploiting SAT Modulo Monotonic Theories (SMMT) solvers. Our tool MsATL combines SMMT solvers with two ATL model checkers: MCMAS and STV. This is the first ever tool for checking the satisfiability of imperfect information ATL. The experimental results show that our approach is quite efficient and quickly checks the satisfiability of large ATL formulae being out of reach of the existing approaches.

1 Introduction

The problem of synthesis is a very important issue in the rapidly-growing field of artificial intelligence and modern software engineering (Jones et al. 2012; Kouvaros, Lomuscio, and Pirovano 2018; Schewe and Finkbeiner 2007). The aim is to automatically develop highly innovative software, also for AI robots, chatbots or autonomous self-driving vehicles. The problem consists in finding a model satisfying a given property, provided the property is satisfiable. Finally, the model is transformed into its correct implementation.

A convenient formalism to specify the game-like interaction between processes in distributed systems is Alternating-Time Temporal Logic (ATL) (Alur, Henzinger, and Kupferman 1997). The interpretation of ATL formulae uses the paradigm of multi-agent systems (MAS) and is defined in models like concurrent game structures or interpreted systems. This logic was introduced to reason about the strategic abilities of agents and their groups. The strategic modalities allow for expressing the ability of agents to force their preferences or to achieve a desired goal and are therefore suitable for describing properties like the existence of a winning strategy. This is particularly important when we study properties and verify the correctness of security protocols or voting systems. There are a lot of papers analysing different versions of ATL (Belardinelli et al. 2019a; Dima and Tiplea 2011; Schobbens 2004; Bulling, Dix, and Jamroga 2010; Dima, Maubert, and Pinchinat 2014; Jamroga, Knapik, and Kurpiewski 2017) and other modal logics of strategic ability (Chatterjee, Henzinger, and Piterman 2010; Mogavero et al. 2012a; Mogavero et al. 2012b). However, there is still a need for developing and introducing new and innovative techniques for solving synthesis and satisfiability problems (Bloem, Könighofer, and Seidl 2014; Bloem et al. 2012; Finkbeiner and Schewe 2013; Kupferman and Vardi 2005). This is because these problems are hard and their solutions require searching for efficient practical algorithms.

1.1 Contribution

Our contribution consists in: (1) introducing a novel technique for checking the ATL satisfiability by applying, for the first time, SAT Modulo Monotonic Theories solvers, and (2) offering a method and a tool for testing the satisfiability in the class of models that meet given restrictions, for different classes of multi-agent systems and ATL semantics, and for the first time for imperfect information strategies.

1.2 Related Work

The problem we are solving is how to decide whether an ATL formula is satisfiable. We call this decision problem $\mathrm{ATL}_i\mathrm{SAT}$ ($\mathrm{ATL}_I\mathrm{SAT}$) for imperfect (resp. perfect) information semantics of ATL. The complexity of $\mathrm{ATL}_I\mathrm{SAT}$ is very high (as discussed below) and the complexity of $\mathrm{ATL}_i\mathrm{SAT}$ is unknown, which makes non-symbolic approaches inefficient. The complexity of $\mathrm{ATL}_I\mathrm{SAT}$ was first proved to be EXPTIME-complete (van Drimmelen 2003; Goranko and Drimmelen 2006) for a fixed number of agents and later extended to the general case in (Walther et al. 2006). The satisfiability of perfect information ATL^*, a generalisation of perfect information ATL, was proved to be 2EXPTIME-complete (Schewe 2008). These results employ alternating tree automata - based techniques. A practically implementable tableau-based constructive decision method for $\mathrm{ATL}_I\mathrm{SAT}$ was described in (Goranko and Shkatov 2009). Subsequently, the tableau-based method was ex-

The work of M. Kacprzak was supported by the Bialystok University of Technology, as part of the research grant WZ/WI-IIT/1/2019 of the Faculty of Computer Science and funded by Ministry of Science and Higher Education, Poland. W. Penczek acknowledges support from Luxembourg/Polish FNR/NCBiR project STV and CNRS/PAS project PARTIES.

tended for checking ATL* (David 2015) and ATEL (Belardinelli 2014), an epistemic extension of ATL. The complexity (and even decidability) of $\mathrm{ATL}_i\mathrm{SAT}$ is unknown. Perhaps a hint of its difficulty is given in (Schobbens 2003; Jamroga and Dix 2006), where it is shown that the problem of model checking of ATL with imperfect information is Δ_2^P-complete but the logic has no standard fixed-point characterisation (Bulling and Jamroga 2011; Dima, Maubert, and Pinchinat 2015).

The previous existing solutions are applicable only to perfect information ATL. Our tool is dedicated for both variants of ATL. For this purpose, we adopt the method based on SAT Modulo Monotonic Theories (SMMT) used to search for models of CTL formulae. This technique was introduced in (Bayless et al. 2015) for building efficient lazy SMT solvers for Boolean monotonic theories. Next, the SMMT framework was used to build an SMT solver for CTL model checking theory and applied to perform efficient and scalable CTL synthesis (Klenze, Bayless, and Hu 2016). In this paper we go one step further by developing an SMMT solver for ATL formulae and show how to construct, often minimal, models for them.

We compare our experimental results with results of the TATL tool (David 2015). Due to the lack of benchmarks for ATL we compare runtimes on random formulas only. TATL implements a tableau-based decision procedure and is the first tool for deciding satisfiability of ATL*. The formulas of the logic are interpreted in Concurrent Game Models (CGM) with perfect recall and perfect information semantics. The main idea behind TATL is to build, step by step, from an initial formula ϕ, a rooted directed graph from which it is possible to extract a CGM satisfying ϕ. The other tool for testing satisfiability of formulae expressed in several logics is Coalgebraic Ontology Logic Reasoner (COOL) (Hausmann, Schröder, and Egger 2016), which allows verification of alternation-free fragments of relational, monotone, and alternating-time μ-calculi and implements a tableau-based global caching algorithm. However, as to the multi-agent systems, COOL supports only Coalition Logic (Pauly 2002) which lacks the temporal dimension and therefore does not allow for inference about the system dynamics – the aspect which is the most interesting for us. The main advantage of our framework consists in promising preliminary experimental results and the fact that we can test $\mathrm{ATL}_i\mathrm{SAT}$ and $\mathrm{ATL}_I\mathrm{SAT}$ in classes of models under given restrictions. Restrictions on the number of agents and their local states for $\mathrm{ATL}_I\mathrm{SAT}$ result directly from the finite model property (Goranko and Drimmelen 2006). The analogous result for $\mathrm{ATL}_i\mathrm{SAT}$ is unknown.

Strategy Logic (SL) extends ATL, and express relevant concepts such as Nash equilibria or Pareto optimality (Belardinelli et al. 2020). Several semantical subclasses of MAS has been identified for which model checking for SL is decidable. The efficient verification algorithms have been developed for: One-Goal SL (Cermák, Lomuscio, and Murano 2015); SL with Simple Goals, where goals are restricted to simple LTL formulae (Belardinelli et al. 2019b); SL with Knowledge for memoryless MAS with incomplete information (Cermák et al. 2018); SL for MAS with imperfect information and public actions, i.e., systems where all actions are visible to all agents (Belardinelli et al. 2017); imperfect-information SL, where the restriction is to a syntactical class of "hierarchical instances" (Berthon et al. 2017). We plan to extend our tool to deal also with the above formalisms.

1.3 Outline of the Approach

Our goal is to build a model for a given formula φ or to show that no model exists wrt. some class of models. To this end a model is represented symbolically by a set of symbolic variables, encoding its global states, the transition function and the valuation function. Over these variables a predicate is defined which is satisfied only by (binary) values representing the model satisfying φ. So, the problem is translated to searching for these values. The search process starts with introducing a partial valuation function assigning values to selected variables and setting some restrictions on the class of models. Then the searching algorithm extends the partial valuation function until all variables are assigned values. The total valuation function constructed defines the model. If this is not a model of φ, then the algorithm withdraws some of the previous assignments. If the predicate is monotonic, techniques for monotonic theories allow to significantly shorten the searching process and early reject a large number of valuations.

The search for a valuation that satisfies the predicate exploits a lazy SMT (*SAT Modulo Theory*) solver, which combines a SAT solver with a set of theory solvers. The satisfaction-checking process relies on two important techniques provided by theory solvers: (a) *theory propagation* that takes a partial truth assignment M_{par} to the theory atoms and checks if the valuation of any other atoms are implied by that partial assignment, or if the partial assignment constitutes a conflict; (b) *clause learning*, where given a conflict, the theory solver finds a subset of M_{par} sufficient to imply the conflict, which the SAT solver can then in turn negate and store as a learned clause. A limitation of SMT solvers is that the efficient algorithms for deciding satisfiability are available for fully specified inputs, but not for partially specified or symbolic inputs. This is different with monotonic theories and SAT Modulo Monotonic Theories (SMMT) solvers (Bayless et al. 2015; Klenze, Bayless, and Hu 2016). In this case, for a given partial truth assignment two completions can be defined: (a) one in which all the unassigned atoms are assigned to false, call it M_{under}, and (b) one in which they are assigned to true, call it M_{over}. What is more, these completions define a subclass of models. Next, a superset of all states satisfying φ in some model of this subclass, can be determined. If the fixed initial state ι does not belong to this set, then there is no model of φ in this subclass.

Furthermore, the search process is performed by an SMMT solver, which exploits both a SAT solver and an ATL model checker. For $\mathrm{ATL_I}$ (perfect information ATL) we use one of the most popular model checker tools, i.e., MCMAS (Lomuscio, Qu, and Raimondi 2017), as well as our own implementation of fixpoint based model-checker for ATL (Niewiadomski et al. 2020). We use STV - the most recent tool dedicated to $\mathrm{ATL_i}$ (imperfect information ATL) (Kurpiewski, Jamroga, and Knapik 2019; Jamroga et al. 2019).

Due to the upper bound we can - in the worst case - check every model consistent with the requirements. Using SAT-like algorithm we have an NP algorithm combined with a model checking algorithm. So, the complexity of our method can stay in NP, if the model checking algorithm is polynomial ($\mathrm{ATL_I}$), or is higher if the model-checking procedure is more expensive ($\mathrm{ATL_i}$).

In Sec. 2 we define a multi-agent system and its model, and give the syntax and semantics of ATL. Sec. 3 defines Boolean monotonic theory for ATL. In Sec. 4 the approximation algorithm is given and its properties are proved. Sec. 5 introduces the algorithm for deciding ATL satisfiability. Sec. 6 and 7 present experimental results and conclusions.

2 MAS and ATL

Alur et al. introduced the ATL logic taking into account different model compositions of open systems like turn-based, synchronous, asynchronous, with fairness constraints or Moore game structures. In this paper we follow Moore synchronous models (Alur, Henzinger, and Kupferman 2002), i.e., assume that the state space is the product of local state spaces, one for each agent, all agents proceed simultaneously, and each agent chooses its next local state independently of the moves of the others. This class of models allows for an efficient testing of the ATL satisfiability.

2.1 Multi-agent System

We start with defining a multi-agent system following (Alur, Henzinger, and Kupferman 2002; Jamroga et al. 2018).

Definition 1. *A* multi-agent system (MAS) *consists of n agents $\mathcal{A} = \{1, \dots, n\}$*[2]*, where each agent $i \in \mathcal{A}$ is associated with a 5-tuple $AG_i = (L_i, \iota_i, Act_i, P_i, T_i)$ including:*
- *a finite set of* local states $L_i = \{l_i^1, l_i^2, \dots, l_i^{n_i}\}$;
- *an* initial local state $\iota_i \in L_i$;
- *a finite set of* local actions $Act_i = \{\epsilon_i, a_i^1, a_i^2, \dots, a_i^{m_i}\}$;
- *a* local protocol $P_i : L_i \to 2^{Act_i}$ *selecting the actions available at each local state, where $\forall_{l_i \in L_i}\ P_i(l_i) \neq \emptyset$;*
- *a (partial)* local transition function $T_i : L_i \times Act_i \to L_i$ *such that $T_i(l_i, a)$ is defined iff $a \in P_i(l_i)$ and $T_i(l_i, \epsilon_i) = l_i$ whenever $\epsilon_i \in P_i(l_i)$ for each $l_i \in L_i$.*

We consider *synchronous* multi-agent systems, where each *global action* is a n-tuple $(a^1, \dots, a^n)$ with $a^i \in Act_i$, i.e., each agent performs one local action. To describe the interaction between agents, the model for MAS is defined.

Definition 2 (Model). *Let $Act = Act_1 \times \cdots \times Act_n$ be the set of all global actions, $\mathcal{PV}$ be a set of the propositional variables, and MAS be a multi-agent system with n agents. An (induced)* model*, is a 4-tuple $M = (\mathcal{S}t, \iota, T, V)$ with*
- *the set $\mathcal{S}t = L_1 \times \cdots \times L_n$ of the global states,*
- *an initial state $\iota = (\iota_1, \dots, \iota_n) \in \mathcal{S}t$,*
- *the global transition function $T : \mathcal{S}t \times Act \to \mathcal{S}t$, such that $T(s_1, a) = s_2$ iff $T_i(s_1^i, a^i) = s_2^i$ for all $i \in \mathcal{A}$, where for global state $s = (l_1, \dots, l_n)$ we denote the local component of agent i by $s^i = l_i$ and for a global action $a = (a^1, \dots, a^n)$, a^i is the local action of agent i.*
- *the valuation of the states $V : \mathcal{S}t \to 2^{\mathcal{PV}}$.*

[2]The environment component may be added here with no technical difficulty.

We say that action $a \in Act$ is *enabled* at $s \in \mathcal{S}t$ if $T(s, a) = s'$ for some $s' \in \mathcal{S}t$. We assume that at each $s \in \mathcal{S}t$ there exists at least one enabled action, i.e., for all $s \in \mathcal{S}t$ exist $a \in Act$, $s' \in \mathcal{S}t$, such that $T(s, a) = s'$. An infinite sequence of global states and actions $\pi = s_0 a_0 s_1 a_1 s_2 \dots$ is called a *path* if $T(s_i, a_i) = s_{i+1}$ for every $i \geq 0$. Let $Act(\pi) = a_0 a_1 a_2 \dots$ be the sequence of actions in π, and $\pi[i] = s_i$ be the i-th global state of π. $\Pi_M(s)$ denotes the set of all paths in M starting at s.

2.2 Alternating-time Temporal Logic

Alternating-time temporal logic, ATL (Alur, Henzinger, and Kupferman 1997; Alur, Henzinger, and Kupferman 1998; Alur, Henzinger, and Kupferman 2002) generalizes the branching-time temporal logic CTL (Clarke and Emerson 1981) by replacing the path quantifiers E, A with *strategic modalities* $\langle\!\langle \Gamma \rangle\!\rangle$. Informally, $\langle\!\langle \Gamma \rangle\!\rangle \gamma$ expresses that the group of agents Γ has a joint strategy to enforce the temporal property γ. The formulae make use of temporal operators: "X" ("next"), "G" ("always from now on"), U ("strong until").

Definition 3 (Syntax of ATL). *In vanilla* ATL*, every occurrence of a strategic modality is immediately followed by a temporal operator. Formally, the language $\mathcal{L}$ of* ATL *is defined by the following grammar: $\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\!\langle \Gamma \rangle\!\rangle X\, \varphi \mid \langle\!\langle \Gamma \rangle\!\rangle \varphi U \varphi \mid \langle\!\langle \Gamma \rangle\!\rangle G \varphi$.*

Let M be a model. A *strategy* of agent $i \in \mathcal{A}$ in M is a conditional plan that specifies what i is going to do in any potential situation.

In this paper we focus on memoryless perfect and imperfect information strategies. Formally, a *memoryless perfect information strategy for agent i* is a function $\sigma_i \colon \mathcal{S}t \to Act_i$ s.t. $\sigma_i(s) \in P_i(s^i)$ for each $s \in \mathcal{S}t$. A *memoryless imperfect information strategy* additionally satisfies $\sigma_i(s) = \sigma_i(s')$ whenever $s^i = (s')^i$. Following (Schobbens 2004), we refer to the former as *I-strategies*, and to the letter as *i-strategies*. Thus, a perfect information strategy can assign different actions to any two global states, while under imperfect information the agent's choice depends only on the local state of the agent. A *joint strategy* σ_Γ for a coalition $\Gamma \subseteq \mathcal{A}$ is a tuple of strategies, one per agent $i \in \Gamma$. We denote the set of Γ's joint memoryless perfect (resp. imperfect) information strategies by $\Sigma_\Gamma^{\mathrm{I}}$ (resp. $\Sigma_\Gamma^{\mathrm{i}}$). Additionally, let $\sigma_\Gamma = (\sigma_1, \dots, \sigma_k)$ be a joint strategy for $\Gamma = \{i_1, \dots, i_k\}$. For each $s \in \mathcal{S}t$, we define $\sigma_\Gamma(s) := (\sigma_1(s), \dots, \sigma_k(s))$.

Definition 4 (Outcome paths). *Let $Y \in \{\mathrm{I}, \mathrm{i}\}$. The* outcome *of strategy $\sigma_\Gamma \in \Sigma_\Gamma^Y$ in state $s \in \mathcal{S}t$ is the set $out_M(s, \sigma_\Gamma) \subseteq \Pi_M(s)$ s.t. $\pi = s_0 a_0 s_1 a_1 \cdots \in out_M(s, \sigma_\Gamma)$ iff $s_0 = s$ and $\forall i \in \mathbb{N}\ \forall j \in \Gamma$, $a_i^j = \sigma_j(\pi[i])$ for $Y = \mathrm{I}$, and $a_i^j = \sigma_j(\pi[i]^j)$ for $Y = \mathrm{i}$.*

Intuitively, the outcome of a joint strategy σ_Γ in a global state s is the set of all the infinite paths that can occur when in each state of the paths agents (an agent) in Γ execute(s) an action according to σ_Γ and agents (an agent) in $\mathcal{A} \setminus \Gamma$ execute(s) an action following their protocols.

The semantics of ATL parameterised with the strategy type $Y \in \{\mathrm{I}, \mathrm{i}\}$, is defined as follows:

$M, s \models_Y \mathsf{p}$ iff $\mathsf{p} \in V(s)$, for $\mathsf{p} \in \mathcal{PV}$;

$M, s \models_Y \neg\varphi$ iff $M, s \not\models_Y \varphi$;

$M, s \models_Y \varphi_1 \wedge \varphi_2$ iff $M, s \models_Y \varphi_1$ and $M, s \models_Y \varphi_2$;

$M, s \models_Y \langle\langle\Gamma\rangle\rangle X\,\varphi$ iff there is a strategy $\sigma_\Gamma \in \Sigma_\Gamma^Y$ such that $out_M(s, \sigma_\Gamma) \neq \emptyset$ and, for each path $\pi \in out_M(s, \sigma_\Gamma)$, we have $M, \pi \models_Y X\,\varphi$, i.e., $M, \pi[1] \models_Y \varphi$;

$M, s \models_Y \langle\langle\Gamma\rangle\rangle \varphi_1 U \varphi_2$ iff there is a strategy $\sigma_\Gamma \in \Sigma_\Gamma^Y$ such that $out_M(s, \sigma_\Gamma) \neq \emptyset$ and, for each path $\pi \in out_M(s, \sigma_\Gamma)$, we have $M, \pi \models_Y \varphi_1 U \varphi_2$, i.e., $M, \pi[i] \models_Y \varphi_2$ for some $i \geq 0$ and $M, \pi[j] \models_Y \varphi_1$ for all $0 \leq j < i$;

$M, s \models_Y \langle\langle\Gamma\rangle\rangle G\varphi$ iff there is a strategy $\sigma_\Gamma \in \Sigma_\Gamma^Y$ such that $out_M(s, \sigma_\Gamma) \neq \emptyset$ and, for each path $\pi \in out_M(s, \sigma_\Gamma)$, we have $M, \pi \models_Y G\varphi$, i.e., $M, \pi[i] \models_Y \varphi$, for every $i \geq 0$.

Definition 5 (Validity). *An* ATL *formula φ is Y-valid in M (denoted $M \models_Y \varphi$) iff $M, \iota \models_Y \varphi$, i.e., φ is true at the initial state of the model M under the Y-strategies.*

By $\mathrm{ATL_I}$ and $\mathrm{ATL_i}$ we denote ATL with the semantics type $Y = \mathrm{I}$ and $Y = \mathrm{i}$, respectively.

The problem we consider is $\mathrm{ATL}_Y\mathrm{SAT}$ with restrictions on models, for each $Y \in \{I, i\}$.

3 Boolean Monotonic Theory for ATL

In this section we show how to construct a Boolean monotonic theory for $\mathrm{ATL_I}$ and $\mathrm{ATL_i}$ – the foundations of a tool for testing the satisfiability of the ATL formulae and for performing efficient and scalable synthesis.

3.1 Boolean Monotonic Theory

Consider a predicate $P : \{0,1\}^n \mapsto \{0,1\}$. We say that P is Boolean *positive monotonic* iff $P(s_1, \ldots, s_{i-1}, 0, s_{i+1}, \ldots, s_n) = 1$ implies $P(s_1, \ldots, s_{i-1}, 1, s_{i+1}, \ldots, s_n) = 1$, for all $1 \leq i \leq n$. P is called Boolean *negative monotonic* iff $P(s_1, \ldots, s_{i-1}, 1, s_{i+1}, \ldots, s_n) = 1$ implies $P(s_1, \ldots, s_{i-1}, 0, s_{i+1}, \ldots, s_n) = 1$, for all $1 \leq i \leq n$.

The definition of (positive and negative Boolean) monotonicity for a function $F : \{0,1\}^n \mapsto 2^S$ (for some set S) is analogous. F is Boolean *positive monotonic* iff $F(s_1, \ldots, s_{i-1}, 0, s_{i+1}, \ldots, s_n) \subseteq F(s_1, \ldots, s_{i-1}, 1, s_{i+1}, \ldots, s_n)$, for all $1 \leq i \leq n$. A function F is Boolean *negative monotonic* iff $F(s_1, \ldots, s_{i-1}, 1, s_{i+1}, \ldots, s_n) \subseteq F(s_1, \ldots, s_{i-1}, 0, s_{i+1}, \ldots, s_n)$, for all $1 \leq i \leq n$.

Definition 6 (Boolean Monotonic Theory). *A theory T with a signature $\Omega = (S, S_f, S_r, ar)$, where S is a non-empty set of elements called sorts, S_f is a set of function symbols, S_r is a set of relation symbols, and ar is the arity of the relation and function symbols, is (Boolean) monotonic iff the only sort in Ω is Boolean, and all predicates and functions in Ω are monotonic.*

3.2 Boolean Encoding of ATL Models

First, we make some assumptions about MAS. Given a set of agents $\mathcal{A} = \{1, \ldots, n\}$, where each agent $i \in \mathcal{A}$ has a fixed set of the local states $L_i = \{l_i^1, \ldots, l_i^{n_i}\}$ and a fixed initial local state $\iota_i \in L_i$. Since agent i can be in one of its n_i local states, and a local transition function T_i is restricted such that it does not involve actions of the other agents, we can assume, without a loss of generality, that agent i has exactly n_i possible actions, i.e., from each local state it can potentially move to each of its local states. So, assume that the set of local actions for agent i is $Act_i = \{a_i^1, \ldots, a_i^{n_i}\}$ and an action a_i^j can move the agent i from any local state to local state l_i^j. Moreover, we assume that each local protocol P_i satisfies that at least one action is available at each local state. Thus, the local transition function T_i for agent i is defined as follows: $T_i(l_i^k, a_i^j) = l_i^j$ if $a_i^j \in P_i(l_i^k)$, for any $l_i^k \in L_i$ and $1 \leq j \leq n_i$.

Next, we represent every single agent i with a given $AG_i = (L_i, \iota_i, Act_i, P_i, T_i)$ by means of a bit vector. In fact, under the assumptions discussed above, we have to encode a local protocol P_i only. It can be defined by a Boolean table lp_i of $|L_i| \times |Act_i|$ entries, where 0 at position (l_i^k, a_i^j) means that the local action a_i^j is not available at the local state l_i^k, and 1 stands for the availability. This table can be represented by a bit vector $tb_i = (lp_i[1], \ldots, lp_i[n_i])$[3], where $lp_i[j]$ stands for the j-th row of the table lp_i, encoding which local actions are available at which local states.

Since the model $M = (\mathcal{S}t, \iota, T, V)$ induced by a MAS is a product of AG_i for $i \in \mathcal{A}$, the bit vector $(tb_1, \ldots, tb_n)$ determines the synchronous product of the local transition functions and thus the global transition function T of M.

Finally, we need to define a valuation of the propositional variables. Given a set $\mathcal{PV}$, a Boolean table of size $|\mathcal{S}t| \times |\mathcal{PV}|$ saves which propositional variables are true in which global states. Then, let $vb = (vb_1, \ldots, vb_k)$ be a bit vector, where $k = |\mathcal{S}t| \cdot |\mathcal{PV}|$, controlling which propositional variables hold in each global state.

In this way, every model can be represented with a bit vector. For a fixed number $|\mathcal{PV}|$ of the propositional variables, a fixed number n of agents, a fixed number n_i of the local states of agent i, for every $i = 1, \ldots, n$, the bit vector $v_M = (tb_1, \ldots, tb_n, vb)$ encodes some model induced by MAS without an initial state fixed. Therefore, v_M actually encodes a family of models differing only in the initial state.

3.3 Predicate *Model*

From now on, we consider models M defined over the fixed number $|\mathcal{PV}|$ of the propositional variables and a fixed number n of agents with fixed numbers $|L_1|, \ldots, |L_n|$ of local states. Thus, we consider models that can be represented by a bit vector v_M consisting of exactly $n_M = |L_1|^2 + \ldots + |L_n|^2 + |\mathcal{S}t| \cdot |\mathcal{PV}|$ bits. In the rest of the work we will use the following notation: $V_m = (TB_1, \ldots, TB_n, VB)$ to denote a vector of Boolean variables, where for $i = 1, \ldots, n$,

[3]In what follows, we assume that a sequence of bit vectors is identified with the bit vector composed of its elements.

TB_i is a vector of $|L_i|^2$ variables and VB is a vector of $|\mathcal{S}t| \cdot |\mathcal{PV}|$ variables.

In order to find a model for a given formula, we apply a symbolic method that consists in finding the Boolean vector encoding this model. To this end, for a fixed $\phi \in \mathcal{L}$, $g \in \mathcal{S}t$, and $Y \in \{\mathrm{I}, \mathrm{i}\}$ the predicate $Model^Y_{g,\phi}(V_m)$ is defined such that for the bit vector v_M,

$$Model^Y_{g,\phi}(v_M) = 1 \text{ if and only if } M, g \models_Y \phi.$$

The search for v_M can be optimized if we use techniques for monotonic theories. Unfortunately, it turns out that the above predicate is not monotonic.

Theorem 1. *The predicate $Model^Y_{g,\phi}(V_m)$ is neither positive nor negative monotonic w.r.t. V_m, for $Y \in \{\mathrm{I}, \mathrm{i}\}$.*

Proof. It follows from a similar result for CTL (Klenze, Bayless, and Hu 2016) as ATL$_\mathrm{I}$ (ATL$_\mathrm{i}$) subsumes CTL. □

However, monotonicity holds for simple subformulae (see Theorem 2). This fact is used to deal with nested formulae by recursively breaking them down into simple ones. Thus, under- and over-approximations (see Sec. 4.1) can be used instead of enumerating and checking all possible models.

Moreover, a new predicate $MApp^Y_{g,\phi}(V_{m_1}, V_{m_2})$, for $Y \in \{\mathrm{I}, \mathrm{i}\}$, defined over two vectors, is introduced (see Sec. 5.1). This predicate is positive monotonic wrt. V_{m_1} and negative monotonic wrt. V_{m_2}, and has the property that for every vector v_M, $MApp^Y_{g,\phi}(v_M, v_M) = Model^Y_{g,\phi}(v_M)$. To compute $MApp^Y_{g,\phi}(v_1, v_2)$ for vectors v_1, v_2 s.t. $v_2[j] \leq v_1[j]$ for every j, the algorithm $SApp^Y$ is used (see Sec. 4.2). It takes as an input a formula ϕ and a partial model M_{par} (see Sec. 4.1), and determines an approximation of a set of states satisfying ϕ in a model M which extends M_{par}. This means that if a state ι does not belong to the approximation, then it cannot satisfy ϕ in any extension of M_{par}. The property allows for using a technique for fast and early elimination of partial evaluations of Boolean variables that cannot be extended to total valuations encoding a model of the formula. The assumption that $v_2[j] \leq v_1[j]$ for every j, ensures that the vectors are not arbitrary, but encode a partial model. The $SApp^Y$ algorithm recursively calculates approximations of subformulae starting from the most nested ones. Each approximation is a set of states that can be represented by a new propositional variable. In this way, we deal with the non-nested formulas at every stage. Moreover, the algorithm extends the input partial model to two total models: $over$ and $under$ and uses a model checker tool to determine the states satisfying subformulae in these models. The correctness of the algorithm follows from the monotonicity properties of the functions: *solve* and MC, which are defined and proved in the next subsection. The proofs are conducted for the ATL formulas under the I semantics. The proofs for the i semantics are omitted since they are exactly the same.

3.4 Functions *solve* and MC

In order to compute the value of the predicate $Model^Y_{g,\phi}(v_M)$ for a given M and $Y \in \{\mathrm{I}, \mathrm{i}\}$, we define a new function, called $solve^Y_\phi(V_m)$. This function returns a set of states of M such that

$$g \in solve^Y_\phi(v_M) \text{ iff } Model^Y_{g,\phi}(v_M) = 1$$

and preserves the following monotonicity properties.

Theorem 2. *The function $solve^Y_\phi(V_m)$, for $Y \in \{\mathrm{I}, \mathrm{i}\}$ is*

1. *positive monotonic w.r.t. VB for $\phi \in \{p, p \wedge q, \langle\langle\Gamma\rangle\rangle X\, p, \langle\langle\Gamma\rangle\rangle Gp, \langle\langle\Gamma\rangle\rangle pUq\}$,*
2. *negative monotonic w.r.t. VB for $\phi = \neg p$,*
3. *positive and negative monotonic w.r.t. TB_i for $i \in \mathcal{A}$ if $\phi \in \{p, \neg p, p \wedge q\}$,*
4. *positive monotonic w.r.t. TB_i for each $i \in \Gamma$ if $\phi \in \{\langle\langle\Gamma\rangle\rangle X\, p, \langle\langle\Gamma\rangle\rangle Gp, \langle\langle\Gamma\rangle\rangle pUq\}$,*
5. *negative monotonic w.r.t. TB_i for $i \in \mathcal{A} \setminus \Gamma$ if $\phi \in \{\langle\langle\Gamma\rangle\rangle X\, p, \langle\langle\Gamma\rangle\rangle Gp, \langle\langle\Gamma\rangle\rangle pUq\}$, where $p, q \in \mathcal{PV}$, $\Gamma \subseteq \mathcal{A}$.*

Regardless of the strategy definition, the above properties hold for perfect and imperfect information semantics what results from the following observations. Adding a local transition for some agent $A \in \Gamma$, increases its number of local choices, but the existing strategies remain. Similarly, adding more states where $p \in \mathcal{PV}$ holds does not affect pre-existing strategies involving p. Moreover, deleting a local transition for agent $B \notin \Gamma$ can reduce only a number of outcome paths of a strategy σ_Γ. Notice also that adding or removing local transitions does not alter the truthfulness of propositional formulae. Similarly, adding more states satisfying p does not affect propositional formulae involving p and reducing the number of states satisfying p does not affect propositional formulae involving $\neg p$.

To compute $solve^Y_\phi(V_m)$ for a formula ϕ and $Y \in \{\mathrm{I}, \mathrm{i}\}$, an evaluation function $MC^Y(op, Z_1, V_m)$ is defined for an unary operator *op* and $MC^Y(op, Z_1, Z_2, V_m)$ for a binary operator *op*, and $Z_1, Z_2 \subseteq \mathcal{S}t$. This function evaluates the operator *op* on sets of states Z_1, Z_2 instead of the formulae holding in these states. If $\phi = p \in \mathcal{PV}$, then for a given model M, $solve^Y_p(v_M)$ returns the set of states of M in which p holds. Otherwise, $solve^Y_\phi(v_M)$ takes the topmost operator *op* of ϕ and solves its argument(s) recursively using the function MC^Y and applying $MC^Y(op, Z_1, v_M)$ or $MC^Y(op, Z_1, Z_2, v_M)$ to the returned set(s) of states. Notice that $MC^Y(\neg, Z, v_M)$ computes the compliment of Z in M, i.e. $\mathcal{S}t \setminus Z$. Similarly, $MC^Y(\wedge, Z_1, Z_2, v_M)$ computes the intersection of Z_1 and Z_2 in M. If $op \in \{\langle\langle\Gamma\rangle\rangle X, \langle\langle\Gamma\rangle\rangle G, \langle\langle\Gamma\rangle\rangle U\}$ the model checking algorithms are used. Now, Theorem 2 can be rewritten by replacing the propositional variables by the sets of states satisfying them.

Theorem 3. *The functions $MC^Y(op, Z_1, V_m)$ for an unary operator op and $MC^Y(op, Z_1, Z_2, V_m)$ for a binary operator op, for $Y \in \{\mathrm{I}, \mathrm{i}\}$ are: 1. positive monotonic w.r.t. VB for $op \in \{\wedge, \langle\langle\Gamma\rangle\rangle X, \langle\langle\Gamma\rangle\rangle G, \langle\langle\Gamma\rangle\rangle U\}$; 2. negative monotonic w.r.t. VB for $op = \neg$; 3. positive and negative monotonic w.r.t. TB_i for $i \in \mathcal{A}$ and $op \in \{\neg, \wedge\}$; 4. positive monotonic w.r.t. TB_i for $i \in \Gamma$ and $op \in \{\langle\langle\Gamma\rangle\rangle X, \langle\langle\Gamma\rangle\rangle G, \langle\langle\Gamma\rangle\rangle U\}$; 5. negative monotonic w.r.t. TB_i for $i \in \mathcal{A} \setminus \Gamma$ and $op \in \{\langle\langle\Gamma\rangle\rangle X, \langle\langle\Gamma\rangle\rangle G, \langle\langle\Gamma\rangle\rangle U\}$.*

4 Approximating ATL Models

This section shows how to approximate models for ATL using SMMT. We start with defining a partial model and *over* and *under* approximations. These approximations determine a class of all models extending the partial model to a total one. Then, the approximation algorithm is introduced and its properties are proven. The algorithm computes over-approximation of the set of all states satisfying the given formula in some model of the class.

4.1 Construction of M_{over} and M_{under}

Given a set of agents $\mathcal{A} = \{1, \dots, n\}$, following Def. 1 we define a *partial protocol* function: $CP_i : L_i \times Act_i \to \{0, 1, undef\}$. By a *partial MAS*, denoted MAS_{CP}, we mean a MAS in which each agent is associated with a partial protocol rather than with a protocol. Then, a model induced by MAS_{CP} together with a *partial valuation of the propositional variables* $CV : \mathcal{S}t \times \mathcal{PV} \to \{0, 1, undef\}$ is called a *partial model*, denoted by M_{par}. Both a partial protocol and a partial valuation, defined to give requirements on the models, can be extended to total functions.

For each partial model M_{par}, total models M^{Γ}_{under} and M^{Γ}_{over}, for $\Gamma \subseteq \mathcal{A}$, are constructed. First, for every agent $i \in \mathcal{A}$ we define: *a necessary local protocol* $\underline{P_i} : L_i \to 2^{Act_i}$ and *a possible local protocol* $\overline{P_i} : L_i \to 2^{Act_i}$, where:
(1) if $CP_i(l_i, a_i) = 1$ then $a_i \in \underline{P_i}(l_i)$ and $a_i \in \overline{P_i}(l_i)$,
(2) if $CP_i(l_i, a_i) = 0$ then $a_i \notin \underline{P_i}(l_i)$ and $a_i \notin \overline{P_i}(l_i)$,
(3) if $CP_i(l_i, a_i) = undef$ then $a_i \notin \underline{P_i}(l_i)$ and $a_i \in \overline{P_i}(l_i)$.
Notice that the possible local protocol is an extension of the necessary local protocol, i.e., for every local state l_i, $\underline{P_i}(l_i) \subseteq \overline{P_i}(l_i)$. Similarly, total valuations of the propositional variables are defined: *a necessary valuation* $\underline{V} : \mathcal{S}t \to 2^{\mathcal{PV}}$ and *a possible valuation* $\overline{V} : \mathcal{S}t \to 2^{\mathcal{PV}}$ such that:
(1) if $CV(g, p) = 1$ then $p \in \underline{V}(g)$ and $p \in \overline{V}(g)$,
(2) if $CV(g, p) = 0$ then $p \notin \underline{V}(g)$ and $p \notin \overline{V}(g)$,
(3) if $CV(g, p) = undef$ then $p \notin \underline{V}(g)$ and $p \in \overline{V}(g)$.
Thus, for every global state $g \in \mathcal{S}t$ we have $\underline{V}(g) \subseteq \overline{V}(g)$.

The total model M^{Γ}_{under} is defined as in Def. 2 of all agents $i \in \Gamma$ with $AG_i = (L_i, \iota_i, Act_i, \underline{P_i}, T_i)$, agents $j \in \mathcal{A} \setminus \Gamma$ with $AG_j = (L_j, \iota_j, Act_j, \overline{P_j}, T_j)$, and for the valuation of the propositional variables $\underline{V}$. The total model M^{Γ}_{over} is defined as in Def. 2 of all agents $i \in \Gamma$ with $AG_i = (L_i, \iota_i, Act_i, \overline{P_i}, T_i)$, agents $j \in \mathcal{A} \setminus \Gamma$ with $AG_j = (L_j, \iota_j, Act_j, \underline{P_j}, T_j)$, and for the valuation of the propositional variables $\overline{V}$.

Definition 7. *Let $M = (\mathcal{S}t, \iota, T, CV)$ and $M' = (\mathcal{S}t, \iota, T, CV')$ be (partial) models defined over $\mathcal{PV}$ and induced by agents $\mathcal{A}$ with $AG_i = (L_i, \iota_i, Act_i, CP_i, T_i)$ and $AG'_i = (L_i, \iota_i, Act_i, CP'_i, T_i)$ for $i \in \mathcal{A}$, resp. We say that M is compatible with M' if M extends M', that is:*

- *(1) if $CP'_i(l_i, a_i) = 1$ then $CP_i(l_i, a_i) = 1$,*
  (2) if $CP'_i(l_i, a_i) = 0$ then $CP_i(l_i, a_i) = 0$,
- *(1) if $CV'(g, p) = 1$ then $CV(g, p) = 1$,*
  (2) if $CV'(g, p) = 0$ then $CV(g, p) = 0$.

Observe that M^{Γ}_{under} and M^{Γ}_{over} are compatible with M_{par} and the vectors $v_{(M)^{\Gamma}_{under}}$ and $v_{(M)^{\Gamma}_{over}}$ uniquely determine M_{par}. Specifically, the elements of a model which are defined in M_{par} are encoded by Boolean values for which $v_{(M)^{\Gamma}_{under}}[j] = v_{(M)^{\Gamma}_{over}}[j]$, and for all other values $v_{(M)^{\Gamma}_{under}}[j] \neq v_{(M)^{\Gamma}_{over}}[j]$.

Theorem 4. *Let a (partial) model M be compatible with a (partial) model M' defined over the same agents and propositional variables, and $\Gamma \subseteq \mathcal{A}$. Then, we have:*

- $v_{(M')^{\Gamma}_{under}}[TB_i] \leq v_{M^{\Gamma}_{under}}[TB_i] \leq v_{M^{\Gamma}_{over}}[TB_i] \leq v_{(M')^{\Gamma}_{over}}[TB_i]$ *for $i \in \Gamma$, and*
- $v_{(M')^{\Gamma}_{under}}[TB_i] \geq v_{M^{\Gamma}_{under}}[TB_i] \geq v_{M^{\Gamma}_{over}}[TB_i] \geq v_{(M')^{\Gamma}_{over}}[TB_i]$ *for $i \in \mathcal{A} \setminus \Gamma$, and*
- $v_{(M')^{\Gamma}_{under}}[VB] \leq v_{M^{\Gamma}_{under}}[VB] \leq v_{M^{\Gamma}_{over}}[VB] \leq v_{(M')^{\Gamma}_{over}}[VB]$,

where $v_M[TB_i]$ is a short for $v_M[tb_i[j_i]]$ for all $1 \leq j_i \leq n_i$ and $v_M[VB]$ is a short for $v_M[vb_j]$ for all $1 \leq j \leq k$.

Proof. Follows from the construction of *over* and *under* models. □

4.2 Algorithm $SApp$

The heart of the approach is Algorithm 1. For a given strategy type $Y \in \{\mathrm{I}, \mathrm{i}\}$, it has on input: an ATL formula ϕ, a (partial) model M_{par}, and λ parameter, which can have one of two values *over* or *under*. The algorithm returns a set of states, which for $\lambda = under$ specifies under-approximation of a set of states satisfying ϕ and for $\lambda = over$ determines the over-approximation of this set. If $\phi = p$ is a propositional variable then $SApp^Y(\phi, M, \lambda)$ computes a set of states satisfying p in the total model $M^{\mathcal{A}}_{\lambda}$. If $\phi = \neg\varphi$, then $SApp^Y(\phi, M, \lambda)$ computes the approximation for a subformula φ performing $SApp^Y(\varphi, M, \lambda')$, where $\lambda' = under$ if $\lambda = over$, and otherwise. Next, the function MC^Y determines the compliment of the set $SApp^Y(\varphi, M, \lambda')$ in $M^{\mathcal{A}}_{\lambda'}$. If $\phi = \langle\langle\Gamma\rangle\rangle X\,\varphi$, then $SApp^Y(\phi, M, \lambda)$ computes the approximation for a subformula φ performing $Z = SApp^Y(\varphi, M, \lambda)$ and the function MC^Y determines the set of states satisfying $\langle\langle\Gamma\rangle\rangle X\,(\kappa_Z)$ in M^{Γ}_{λ}, where κ_Z is a propositional formula satisfied only by states belonging to Z. In fact, MC^Y has a set of states as its input, not a formula. This technique allows calculations of approximations for formulas with nesting of various coalition operators. For other operators, the algorithm works similarly.

Theorem 5. *Let M_{par} and M'_{par} be two (partial) models s.t. M_{par} is compatible with M'_{par}. Then, for $Y \in \{\mathrm{I}, \mathrm{i}\}$, $SApp^Y(\phi, M_{par}, over) \subseteq SApp^Y(\phi, M'_{par}, over)$ and $SApp^Y(\phi, M'_{par}, under) \subseteq SApp^Y(\phi, M_{par}, under)$.*

Proof. By a structural induction on a formula ϕ. It is the same for both semantics: $Y = \mathrm{I}$ and $Y = \mathrm{i}$. We show the first case. Since M_{par} is compatible with M'_{par} then $v_{M^{\Gamma}_{over}}[j] \leq v_{(M')^{\Gamma}_{over}}[j]$ and $v_{(M')^{\Gamma}_{under}}[j] \leq v_{M^{\Gamma}_{under}}[j]$ for any Γ and $j \in \{VB, TB_i\}$ with $i \in \Gamma$ and $v_{M^{\Gamma}_{over}}[j] \geq v_{(M')^{\Gamma}_{over}}[j]$ and $v_{(M')^{\Gamma}_{under}}[j] \geq v_{M^{\Gamma}_{under}}[j]$ for any Γ and $j = TB_i$ with $i \in \mathcal{A} \setminus \Gamma$.
The base case. If $\phi = p \in \mathcal{PV}$, then from the definition of the algorithm, $SApp^I(p, M_{par}, over)$ returns the set of the states satisfying p in $M^{\mathcal{A}}_{over}$ and $SApp^I(p, (M')_{par}, over)$

Algorithm 1 Algorithm $SApp^Y$

```
Input: $\phi, M_{par}, \lambda$
Output: a set of states
1: if $\phi \in \mathcal{PV}$ then
2:     return $\{g \in \mathcal{S}t : M^{\mathcal{A}}_{\lambda}, g \models_Y \phi\}$
3: else if $\phi = op(\psi)$ then
4:     if op is $\neg$ then    // negative monotonic
5:         if $\lambda = over$ then
6:             $Z := SApp^Y(\psi, M_{par}, under)$
7:             return $MC^Y(op, Z, v_{M^{\mathcal{A}}_{under}})$
8:         else
9:             $Z := SApp^Y(\psi, M_{par}, over)$
10:            return $MC^Y(op, Z, v_{M^{\mathcal{A}}_{over}})$
11:    else    // op $\in \{\langle\langle\Gamma\rangle\rangle X, \langle\langle\Gamma\rangle\rangle G\}$
12:        $Z := SApp^Y(\psi, M_{par}, \lambda)$
13:        return $MC^Y(op, Z, v_{M^{\Gamma}_{\lambda}})$
14: else if $\phi = op(\psi_1, \psi_2)$ for $op \in \{\langle\langle\Gamma\rangle\rangle U, \wedge\}$ then
15:    $Z_1 := SApp^Y(\psi_1, M_{par}, \lambda)$
16:    $Z_2 := SApp^Y(\psi_2, M_{par}, \lambda)$
17:    if op is $\langle\langle\Gamma\rangle\rangle U$ then
18:        return $MC^Y(op, Z_1, Z_2, v_{M^{\Gamma}_{\lambda}})$
19:    else    // op is $\wedge$
20:        return $MC^Y(op, Z_1, Z_2, v_{M^{\mathcal{A}}_{\lambda}})$
```

returns the set of the states satisfying p in $(M')^{\mathcal{A}}_{over}$. Thus $SApp^I(\phi, M_{par}, over) \subseteq SApp^I(\phi, M'_{par}, over)$ since $v_{M^{\Gamma}_{over}}[VB] \leq v_{(M')^{\Gamma}_{over}}[VB]$. Similarly, $SApp^I(\phi, M'_{par}, under) \subseteq SApp^I(\phi, M_{par}, under)$ since $v_{(M')^{\Gamma}_{under}}[VB] \leq v_{M^{\Gamma}_{under}}[VB]$.

The induction step. We show the proof for the unary operators $\neg$, $\langle\langle\Gamma\rangle\rangle X$, $\langle\langle\Gamma\rangle\rangle G$. The proofs for the binary operators $Until$ and $\wedge$ are similar.

Induction assumption (IA): the thesis holds for a formula ψ. Induction hypothesis (IH): the thesis holds for $\phi = op(\psi)$.

• If $\phi = \neg\psi$.

Let $Z = SApp^I(\psi, M_{par}, under)$ and $Z' = SApp^I(\psi, M'_{par}, under)$, then $Z' \subseteq Z$ from IA. Then, $MC^I(\neg, Z, v_{M^{\mathcal{A}}_{under}}) \subseteq MC^I(\neg, Z', v_{M^{\mathcal{A}}_{under}})$ since MC^I for $op = \neg$ returns the compliment of Z and Z', respectively. Next, $MC^I(\neg, Z', v_{M^{\mathcal{A}}_{under}}) \subseteq MC^I(\neg, Z', v_{(M')^{\mathcal{A}}_{under}})$ since MC^I is negative monotonic w.r.t. VB and TB_i for $i \in \mathcal{A}$ from Theorem 3. Thus, $MC^I(\neg, Z, v_{M^{\mathcal{A}}_{under}}) \subseteq MC^I(\neg, Z', v_{(M')^{\mathcal{A}}_{under}})$ and $SApp^I(\phi, M_{par}, over) \subseteq SApp^I(\phi, M'_{par}, over)$.

Let $Z = SApp^I(\psi, M_{par}, over)$ and $Z' = SApp^I(\psi, M'_{par}, over)$, then $Z \subseteq Z'$ from IA. Then, $MC^I(\neg, Z', v_{(M')^{\mathcal{A}}_{over}}) \subseteq MC^I(\neg, Z, v_{(M')^{\mathcal{A}}_{over}})$ since MC^I for $op = \neg$ returns the compliment of Z' and Z, respectively. Next, $MC^I(\neg, Z, v_{(M')^{\mathcal{A}}_{over}}) \subseteq MC^I(\neg, Z, v_{M^{\mathcal{A}}_{over}})$ since MC^I is negative monotonic w.r.t. VB and TB_i for $i \in \mathcal{A}$ from Theorem 3. Thus, $MC^I(\neg, Z', v_{(M')^{\mathcal{A}}_{over}}) \subseteq MC^I(\neg, Z, v_{M^{\mathcal{A}}_{over}})$ and $SApp^I(\phi, M'_{par}, under) \subseteq SApp^I(\phi, M_{par}, under)$.

• If $\phi = op(\psi)$ with $op \in \langle\langle\Gamma\rangle\rangle X$, $\langle\langle\Gamma\rangle\rangle G$.

Let $Z = SApp^I(\psi, M_{par}, over)$ and $Z' = SApp^I(\psi, M'_{par}, over)$, then $Z \subseteq Z'$ from IA. Observe that $SApp^I(\phi, M_{par}, over)$ is $MC^I(op, Z, v_{M^{\Gamma}_{over}})$. Similarly, $SApp^I(\phi, M'_{par}, over)$ is $MC^I(op, Z', v_{(M')^{\Gamma}_{over}})$. Since $Z \subseteq Z'$ then $MC^I(op, Z, v_{M^{\Gamma}_{over}}) \subseteq MC^I(op, Z', v_{M^{\Gamma}_{over}})$. Next $MC^I(op, Z', v_{M^{\Gamma}_{over}}) \subseteq MC^I(op, Z', v_{(M')^{\Gamma}_{over}})$ since MC for $op \in \langle\langle\Gamma\rangle\rangle X$, $\langle\langle\Gamma\rangle\rangle G$ is positive monotonic w.r.t. VB and TB_i for $i \in \Gamma$ and negative monotonic w.r.t. TB_i for $i \in \mathcal{A} \setminus \Gamma$ from Theorem 3. Finally $MC^I(op, Z, v_{M^{\Gamma}_{over}}) \subseteq MC^I(op, Z', v_{(M')^{\Gamma}_{over}})$ and $SApp^I(\phi, M_{par}, over) \subseteq SApp^I(\phi, M'_{par}, over)$.

Let $Z = SApp^I(\psi, M_{par}, under)$ and $Z' = SApp^I(\psi, M'_{par}, under)$, then $Z' \subseteq Z$ from IA. Observe that $SApp^I(\phi, M_{par}, under)$ is $MC^I(op, Z, v_{M^{\Gamma}_{under}})$. Similarly, $SApp^I(\phi, M'_{par}, over)$ is $MC^I(op, Z', v_{(M')^{\Gamma}_{under}})$. Since $Z' \subseteq Z$ then $MC^I(op, Z', v_{(M')^{\Gamma}_{under}}) \subseteq MC^I(op, Z, v_{(M')^{\Gamma}_{under}})$. Next $MC^I(op, Z, v_{(M')^{\Gamma}_{under}}) \subseteq MC^I(op, Z, v_{M^{\Gamma}_{under}})$ since MC for $op \in \langle\langle\Gamma\rangle\rangle X$, $\langle\langle\Gamma\rangle\rangle G$ is positive monotonic w.r.t. VB and TB_i for $i \in \Gamma$ and negative monotonic w.r.t. TB_i for $i \in \mathcal{A} \setminus \Gamma$ from Theorem 3. Finally $MC^I(op, Z', v_{(M')^{\Gamma}_{under}}) \subseteq MC^I(op, Z, v_{M^{\Gamma}_{under}})$ and $SApp^I(\phi, M'_{par}, under) \subseteq SApp^I(\phi, M_{par}, under)$. □

The algorithm $SApp^Y$, for a total model M compatible with a partial model M_{par}, computes over and under-approximation of $solve^Y_{\phi}(v_M)$, for $Y \in \{\text{I}, \text{i}\}$. More precisely, $SApp^Y(\phi, M_{par}, over)$ returns a set of states, which is an over-approximation of $solve^Y_{\phi}(v_M)$. This means that if $\iota \in solve^Y_{\phi}(v_M)$, then $\iota \in SApp^Y(\phi, M_{par}, over)$. Clearly, if $\iota \notin SApp^Y(\phi, M_{par}, over)$, then there is no model M extending M_{par} such that $M, \iota \models_Y \phi$. Similarly, $SApp^Y(\phi, M_{par}, under)$ computes an under-approximation of $solve^Y_{\phi}(v_M)$. This means that if $\iota \in SApp^Y(\phi, M_{par}, under)$ then $\iota \in solve^Y_{\phi}(v_M)$.

Theorem 6. *Let M_{par} be a partial model and M be a total model compatible with M_{par}. Then, for a formula ϕ we have: $SApp^Y(\phi, M_{par}, under) \subseteq solve^Y_{\phi}(v_M) \subseteq SApp^Y(\phi, M_{par}, over)$, for $Y \in \{\text{I}, \text{i}\}$.*

Proof. Follows from Theorem 5 and the fact that $SApp^Y(\phi, M, under) = SApp^Y(\phi, M, over) = solve^Y_{\phi}(v_M)$, for $Y \in \{\text{I}, \text{i}\}$, since M is total. □

5 Satisfiability Procedure

Basing on the SMMT framework, we have implemented the MsATL tool. Our implementation exploits a slightly modified MiniSAT (Eén and Sörensson 2003) as a SAT-solving core, and $SApp^Y$ algorithm as the theory solver for ATL.

5.1 Monotonic Predicate $MApp$

First, the predicate $MApp^Y_{g,\phi}(V_{m_1}, V_{m_2})$, for $Y \in \{\mathrm{I}, \mathrm{i}\}$, is defined such that $MApp^Y_{g,\phi}(v_1, v_2) = 1$ iff bit vectors v_1 and v_2 determine a (partial) model M_{par}, i.e. $v_2[j] \le v_1[j]$ for all j, and $g \in SApp^Y(\phi, M_{par}, over)$. The predicate is monotonic in the sense that for bit vectors v_1, v_2 and v'_1, v'_2 such that $v'_2[i] \le v_2[i]$ and $v_1[i] \le v'_1[i]$ for all i, if $MApp^Y_{g,\phi}(v_1, v_2) = 1$, then $MApp^Y_{g,\phi}(v'_1, v'_2) = 1$, what follows from Theorem 5.

Constructing a model that satisfies a formula ϕ is therefore reduced to searching for bit vectors v_1, v_2 satisfying predicate $MApp^Y_{g,\phi}(V_{m_1}, V_{m_2})$ and such that $v_1 = v_2$. The searching method starts with two vectors v_1, v_2 such that $v_2[j] \le v_1[j]$ for all j and, step by step, makes decisions about the unification of their values in positions where they differ. If $MApp^Y_{g,\phi}(v_1, v_2) = 1$, then new decisions can be made (with no guarantee of success). If $MApp^Y_{g,\phi}(v_1, v_2) = 0$, then some previous decisions are withdrawn and the process continues. If $v_1 = v_2$ and $MApp^Y_{g,\phi}(v_1, v_2) = 1$, then v_1 represents a model M satisfying ϕ, i.e. $M, g \models_Y \phi$.

5.2 Satisfiability Tool

Below we describe how the algorithm searching for values satisfying $MApp^Y_{g,\phi}(V_{m_1}, V_{m_2})$, on the basis of which the tool solving ATL$_Y$SAT with restrictions, is built. For a fixed $Y \in \{\mathrm{I}, \mathrm{i}\}$ the following input and output are defined.
Input: (a) an ATL formula ϕ, (b) model requirements determining a partial model M_{par}.
Output: a model satisfying ϕ, meeting the requirements of M_{par} or the answer that such a model does not exist.
Steps of the algorithm: Let d be an integer variable tracking the decision depth of the solver, V_m be a vector of Boolean variables over which $MApp^Y_{\iota,\phi}(V_{m_1}, V_{m_2})$ is defined, and $asg(i)$ denote the variable assigned at depth i.
1. Let $d := 0$. Set values of selected variables of V_m according to the M_{par} requirements.
2. Compute $SApp^Y(\phi, M_{par}, over)$.
3. If $\iota \in SApp^Y(\phi, M_{par}, over)$, then (a) if all variables of V_m have assigned values and v is the determined vector of Boolean values, then $MApp^Y_{\iota,\phi}(v, v) = 1$. Return the model represented by v. (b) otherwise: $d := d + 1$, and the SAT-solver assigns a value to the variable $asg(d) \in V_m$. In this way, we have a new partial model M_{par} restricting the class of the considered models. Go to step 2.
4. If $\iota \notin SApp^Y(\phi, M_{par}, over)$, then (a) if $d > 0$, then compute a conflict clause, analyse the conflict, undo recent decisions until appropriate depth c, $d := c$, assign the opposite value to the variable $asg(c)$, and go to step 2. (b) if $d = 0$ there is no model meeting the requirements and satisfying ϕ. Return UNSAT.

Example 1. *Let us consider the formula $\beta = \neg p \wedge \langle\langle 0 \rangle\rangle X\, p$. We want to check if (under perfect information semantics) there is a model for β consisting of two agents with two local states. Thus, the potential global model has 4 global states. We allocate 4 symbolic variables for each agent to represent its local transitions (TB), and 4 to encode the proposition valuation (VB).*

Initially, the SAT solver's assignment to variables of VB is empty, and the assignment to variables of TB includes only the variables representing the self-loops. These are assigned true in order to interpret the formula over infinite paths and to ensure that each agent always can do something at every state. It determines the initial M_{par} as shown in Fig. 1. The valuation of variables of TB_i is shown as a matrix. The values at (s, t) determine if the transition from state s to t is enabled (T true, solid line), disabled (F $false$, nothing), or enabled in overapproximation and disabled in underapproximation (U undefined, dashed line). The colours of global states represent the valuation of VB variables: green - the property p holds, black - does not hold, and yellow - holds in $v_{M^{\mathcal{A}}_{over}}$, and does not hold in $v_{M^{\mathcal{A}}_{under}}$. The double circles denote the initial states.

We start with computing $SApp^I(\beta, M_{par}, over)$. According to Algorithm 1 (line 14) we have:
$Z_1 = SApp^I(\neg p, M_{par}, over) =$
$MC^I(\neg, SApp^I(p, M_{par}, under), v_{M^{\mathcal{A}}_{under}}) =$
$MC^I(\neg, \emptyset, v_{M^{\mathcal{A}}_{under}}) = \{0, 1, 2, 3\}$*, and*
$Z_2 = SApp^I(\langle\langle 0 \rangle\rangle X\, p, M_{par}, over) =$
$MC^I(\langle\langle 0 \rangle\rangle X, SApp^I(p, M_{par}, over), v_{M^{\{0\}}_{over}}) =$
$MC^I(\langle\langle 0 \rangle\rangle X, \{0, 1, 2, 3\}, v_{M^{\{0\}}_{over}}) = \{0, 1, 2, 3\}$[4].
Finally, we have the set: $MC^I(\wedge, Z_1, Z_2, v_{M^{\mathcal{A}}_{over}}) = Z_1 \cap Z_2 = \{0, 1, 2, 3\}$. The initial state 0 belongs to the resulting set, so it is possible that the partial model M_{par} could be extended to a model for the formula. The algorithm continues and the solver assigns a value to some undefined variable. We obtain a new partial model M'_{par} and proceed again with $SApp^I(\beta, M'_{par}, over)$.

Assume, that after several assignments our partial model M_{par} is as in Fig. 2. All variables of TB_1 are assigned, and only one of variables of TB_0 and one of VB are undefined. As before, the algorithm computes $SApp^I(\beta, M_{par}, over)$:
$Z_1 = SApp^I(\neg p, M_{par}, over) =$
$MC^I(\neg, SApp^I(p, M_{par}, under), v_{M^{\mathcal{A}}_{under}}) =$
$MC^I(\neg, \{0, 3\}, v_{M^{\mathcal{A}}_{under}}) = \{1, 2\}$*, and*
$Z_2 = SApp^I(\langle\langle 0 \rangle\rangle X\, p, M_{par}, over) =$
$MC^I(\langle\langle 0 \rangle\rangle X, SApp^I(p, M_{par}, over), v_{M^{\{0\}}_{over}}) =$
$MC^I(\langle\langle 0 \rangle\rangle X, \{0, 2, 3\}, v_{M^{\{0\}}_{over}}) = \{0, 2, 3\}$.
Then we have: $MC^I(\wedge, Z_1, Z_2, v_{M^{\mathcal{A}}_{over}}) = Z_1 \cap Z_2 = \{2\}$. The initial state 0 does not belong to the set, so it is not possible to extend M_{par} to a model for β. Thus, the algorithm backtracks and changes some previous assignments.

Our procedure always finds a model for an ATL formula under given restrictions, provided it exists in the given class. However, if UNSAT is returned, there is no model satisfying the formula under given restrictions, but it does not mean that such a model does not exist at all.

[4] $Z_2 = \{0, 2\}$ if the model-checking procedure considers only the reachable states.

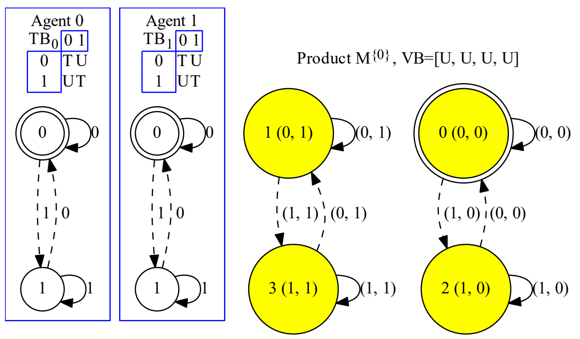

Figure 1: The initial partial model M_{par}

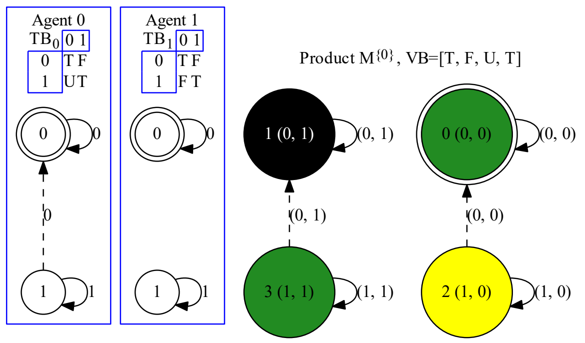

Figure 2: A partial model M_{par} after several assignments

6 Experimental Results

Due to the lack of standard benchmarks for testing the satisfiability of ATL, we have implemented an ATL formula generator which, given a number of agents, groups, and propositional variables, draws a random ATL formula. We have compared our preliminary results for ATL$_I$SAT with TATL (David 2015). Despite the fact that our implementation is at the prototype stage, and there is a lot of space for optimizations, the results are encouraging. Both tools need only fractions of a second to test small formulae. When the formula grows (especially in depth), the time consumed by both tools also quickly increases. Moreover, which is typical for SAT-based methods, MsATL's runtime is higher for unsatisfiable formulae and large state spaces. It can be improved by symmetry reductions preventing the exploration of many isomorphic models. However, we have found a class of formulae for which our tool easily outperforms TATL – large formulae satisfied by relative small models. Table 1 presents the results for such formulae generated for $|\mathcal{PV}| = 3$, $|\mathcal{A}| = 3$, and 4 groups. The table rows contain: a formula id, the number of nested strategy operators, the number of Boolean connectives, and computation times[5].

Due to lack of space we show only[6] Form. 1 of Table 1: $\langle\langle 0\rangle\rangle \mathrm{X}(\neg p_0 \vee \langle\langle 1\rangle\rangle \mathrm{G}(\neg p_1 \vee \langle\langle 0,1\rangle\rangle \mathrm{F}(\neg p_1 \vee \langle\langle 0,1\rangle\rangle \mathrm{F}(\neg p_0 \vee \langle\langle 2\rangle\rangle \mathrm{F}\langle\langle 0\rangle\rangle \mathrm{X}(\neg p_0 \vee \langle\langle 1\rangle\rangle \mathrm{G}(\neg p_1 \vee \langle\langle 0,1\rangle\rangle \mathrm{G}(\langle\langle 0\rangle\rangle \mathrm{F}\neg p_0)))))))$.

[5]The experiments have been performed using a PC equipped with Intel i5-7200U CPU and 16GB RAM running Linux.

[6]Additional resources, including a prototype version of our tool, the benchmarks, can be accessed at http://monosatatl.epizy.com

Id	1	2	3	4	5	6	7	8
Depth	9	13	17	20	23	26	30	33
Con.	13	19	25	31	35	41	49	55
MsAtl[s]	0.22	0.23	0.24	0.31	0.32	0.34	0.38	0.43
TATL[s]	0.58	6.2	29.7	74.6	229	552	1382	3948

Table 1: Experimental results for perfect information.

Id	Gr.	Depth	Con.	L=2	L=3	L=4	L=5
1	1	2	4	12.1	37.2	88.8	226
2	2	3	9	16.4	52.7	167	542
3	3	3	6	15.8	56.6	163	559
4	3	4	6	22.9	68.1	194	746
5	4	7	6	35.8	124	285	795
6	5	13	13	70.9	265	647	2480
7	5	17	15	88.2	314	744	2365
8	5	21	18	106	383	1110	3470

Table 2: Experimental results for imperfect information.

Table 2 presents experimental results for randomly generated formulae of ATL$_\mathrm{i}$, with MsATL calling STV for the model checking subtask. The column 'Gr.' stands for the number of distinct groups of agents in the formula, and the columns marked 'L' contain computation times (sec.) for different numbers of local states per agent. These results are by no means comprehensive, but they show the potential of our method. It is easy to observe, that dealing with imperfect information significantly affects the computation times.

Perfect vs. imperfect information. Satisfiability in perfect information models implies satisfiability for imperfect information, but not vice versa (Bulling and Jamroga 2014). To test MsATL on a (non-randomly generated) case that requires imperfect information, we used formula $\neg\phi$, where $\phi \equiv \big(\neg\mathsf{next} \wedge \langle\langle 1\rangle\rangle F\,\mathsf{next} \wedge \langle\langle \emptyset\rangle\rangle G(\mathsf{next} \rightarrow \langle\langle 1\rangle\rangle F\,\mathsf{win})\big) \rightarrow \langle\langle 1\rangle\rangle F\,\mathsf{win}$. Intuitively, ϕ expresses that, if agent 1 can get to a "next" state, and whenever in "next" it has a follow-up strategy to win, then agent 1 must also have a single strategy to win.[7] Formulae like ϕ are known to be valid for ATL$_\mathrm{I}$ but not for ATL$_\mathrm{i}$ (Bulling and Jamroga 2014). Our tool, using two local states per agent, determined the negation of ϕ to be satisfiable for ATL$_\mathrm{i}$ (80 sec.) and unsatisfiable for ATL$_\mathrm{I}$ (11 sec.), which demonstrates that both functionalities of MsATL are important.

7 Conclusions

The paper introduced a new method exploiting SMMT solvers for (bounded) testing of ATL satisfiability and for constructing (in many cases minimal) ATL models. Despite the fact that we apply the method to a restricted class of models for ATL under the standard semantics, our method can be adapted to other classes of multi-agent systems as well as to other ATL semantics including imperfect information. Although our implementation is at the preliminary stage, the experimental results show a high potential for this approach.

[7]We could not use a more straightforward formalization, since MsATL calls STV for model checking, and STV does not admit the "next time" operator X .

References

Alur, R.; Henzinger, T. A.; and Kupferman, O. 1997. Alternating-time temporal logic. In *Proc. of the 38th IEEE Symp. on Foundations of Computer Science (FOCS'97)*, 100–109. IEEE Computer Society.

Alur, R.; Henzinger, T. A.; and Kupferman, O. 1998. Alternating-time temporal logic. *LNCS* 1536:23–60.

Alur, R.; Henzinger, T. A.; and Kupferman, O. 2002. Alternating-time temporal logic. *Journal of the ACM* 49(5):672–713.

Bayless, S.; Bayless, N.; Hoos, H.; and Hu, A. 2015. SAT modulo monotonic theories. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, 3702–3709. AAAI Press.

Belardinelli, F.; Lomuscio, A.; Murano, A.; and Rubin, S. 2017. Verification of multi-agent systems with imperfect information and public actions. In Larson, K.; Winikoff, M.; Das, S.; and Durfee, E. H., eds., *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, 1268–1276. ACM.

Belardinelli, F.; Lomuscio, A.; Murano, A.; and Rubin, S. 2019a. Imperfect information in alternating-time temporal logic on finite traces. In *PRIMA 2019: Principles and Practice of Multi-Agent Systems - 22nd International Conference, Turin, Italy, October 28-31, 2019, Proceedings*, 469–477.

Belardinelli, F.; Jamroga, W.; Kurpiewski, D.; Malvone, V.; and Murano, A. 2019b. Strategy logic with simple goals: Tractable reasoning about strategies. In Kraus, S., ed., *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 88–94. ijcai.org.

Belardinelli, F.; Lomuscio, A.; Murano, A.; and Rubin, S. 2020. Verification of multi-agent systems with public actions against strategy logic. *Artif. Intell.* 285:103302.

Belardinelli, F. 2014. Reasoning about knowledge and strategies: Epistemic strategy logic. In *Proceedings 2nd International Workshop on Strategic Reasoning, SR 2014, Grenoble, France, April 5-6, 2014*, 27–33.

Berthon, R.; Maubert, B.; Murano, A.; Rubin, S.; and Vardi, M. Y. 2017. Strategy logic with imperfect information. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, 1–12. IEEE Computer Society.

Bloem, R.; Jobstmann, B.; Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2012. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.* 78:911–938.

Bloem, R.; Könighofer, R.; and Seidl, M. 2014. SAT-based synthesis methods for safety specs. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, 1–20. Springer.

Bulling, N., and Jamroga, W. 2011. Alternating epistemic μ-calculus. In *Proceedings of IJCAI-11*, 109–114.

Bulling, N., and Jamroga, W. 2014. 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.

Bulling, N.; Dix, J.; and Jamroga, W. 2010. Model checking logics of strategic ability: Complexity. In Dastani, M.; Hindriks, K.; and Meyer, J.-J., eds., *Specification and Verification of Multi-Agent Systems*. Springer. 125–159.

Cermák, P.; Lomuscio, A.; Mogavero, F.; and Murano, A. 2018. Practical verification of multi-agent systems against SLK specifications. *Inf. Comput.* 261(Part):588–614.

Cermák, P.; Lomuscio, A.; and Murano, A. 2015. Verifying and synthesising multi-agent systems against one-goal strategy logic specifications. In Bonet, B., and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, 2038–2044. AAAI Press.

Chatterjee, K.; Henzinger, T.; and Piterman, N. 2010. Strategy logic. *Inf. Comput.* 208(6):677–693.

Clarke, E., and Emerson, E. 1981. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of Logics of Programs Workshop*, volume 131 of *Lecture Notes in Computer Science*, 52–71.

David, A. 2015. Deciding ATL* satisfiability by tableaux. In *International Conference on Automated Deduction*, 214–228. Springer.

Dima, C., and Tiplea, F. 2011. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR* abs/1102.4225.

Dima, C.; Maubert, B.; and Pinchinat, S. 2014. The expressive power of epistemic μ-calculus. *CoRR* abs/1407.5166.

Dima, C.; Maubert, B.; and Pinchinat, S. 2015. Relating paths in transition systems: The fall of the modal μ-calculus. In *Proceedings of MFCS*, volume 9234 of *Lecture Notes in Computer Science*, 179–191. Springer.

Eén, N., and Sörensson, N. 2003. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, 502–518. Springer.

Finkbeiner, B., and Schewe, S. 2013. Bounded synthesis. *International Journal on Software Tools for Technology Transfer* 15(5-6):519–539.

Goranko, V., and Drimmelen, G. V. 2006. Complete axiomatization and decidability of alternating-time temporal logic. *Theoretical Computer Science* 353(1-3):93–117.

Goranko, V., and Shkatov, D. 2009. Tableau-based decision procedures for logics of strategic ability in multiagent systems. *ACM Trans. Comput. Log.* 11(1):3:1–3:51.

Hausmann, D.; Schröder, L.; and Egger, C. 2016. Global caching for the alternation-free μ-calculus. In Desharnais, J., and Jagadeesan, R., eds., *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, 34:1–34:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Jamroga, W., and Dix, J. 2006. Model checking ATL$_{ir}$ is indeed Δ_2^P-complete. In *Proceedings of EUMAS'06*, volume 223 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Jamroga, W.; Penczek, W.; Dembiński, P.; and Mazurkiewicz, A. 2018. Towards partial order reductions for strategic ability. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, 156–165.

Jamroga, W.; Knapik, M.; Kurpiewski, D.; and Mikulski, Ł. 2019. Approximate verification of strategic abilities under imperfect information. *Artif. Intell.* 277.

Jamroga, W.; Knapik, M.; and Kurpiewski, D. 2017. Fixpoint approximation of strategic abilities under imperfect information. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, 1241–1249.

Jones, A. V.; Knapik, M.; Penczek, W.; and Lomuscio, A. 2012. Group synthesis for parametric temporal-epistemic logic. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, 1107–1114.

Klenze, T.; Bayless, S.; and Hu, A. 2016. Fast, flexible, and minimal CTL synthesis via SMT. In Chaudhuri, S., and Farzan, A., eds., *Computer Aided Verification*, 136–156. Springer International Publishing.

Kouvaros, P.; Lomuscio, A.; and Pirovano, E. 2018. Symbolic synthesis of fault-tolerance ratios in parameterised multi-agent systems. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 324–330.

Kupferman, O., and Vardi, M. 2005. Safraless decision procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, 531–540. IEEE.

Kurpiewski, D.; Jamroga, W.; and Knapik, M. 2019. STV: model checking for strategies under imperfect information. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, 2372–2374.

Lomuscio, A.; Qu, H.; and Raimondi, F. 2017. MCMAS: an open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer* 19(1):9–30.

Mogavero, F.; Murano, A.; Perelli, G.; and Vardi, M. 2012a. A decidable fragment of strategy logic. *CoRR* abs/1202.1309.

Mogavero, F.; Murano, A.; Perelli, G.; and Vardi, M. 2012b. What makes ATL* decidable? A decidable fragment of strategy logic. In *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, 193–208.

Niewiadomski, A.; Kacprzak, M.; Kurpiewski, D.; Knapik, M.; Penczek, W.; and Jamroga, W. 2020. MsATL: A tool for SAT-based ATL satisfiability checking. In Seghrouchni, A. E. F.; Sukthankar, G.; An, B.; and Yorke-Smith, N., eds., *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, 2111–2113. International Foundation for Autonomous Agents and Multiagent Systems.

Pauly, M. 2002. A modal logic for coalitional power in games. *J. Log. Comput.* 12(1):149–166.

Schewe, S., and Finkbeiner, B. 2007. Distributed synthesis for alternating-time logics. In *Automated Technology for Verification and Analysis, 5th International Symposium, ATVA 2007, Tokyo, Japan, October 22-25, 2007, Proceedings*, 268–283.

Schewe, S. 2008. ATL* satisfiability is 2EXPTIME-complete. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, 373–385.

Schobbens, P. Y. 2003. ATL with imperfect recall. Presented at the workshop "Logic and Communication in Multi-Agent Systems", June 29, 2003, Eindhoven, Netherlands.

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

van Drimmelen, G. 2003. Satisfiability in alternating-time temporal logic. In *18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.*, 208–217. IEEE.

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