SAT-Based ATL Satisfiability Checking

Magdalena Kacprzak¹, Artur Niewiadomski², Wojciech Penczek³

¹Bialystok University of Technology, Bialystok, Poland
²Siedlce University, Faculty of Exact and Natural Sciences, Siedlce, Poland
³Institute 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, Knapi, and Kurpiewski 2017) and other modal logics of strategic ability (Chatterjee, Henzinger, and Piterman 2010; Mogavero et al. 2012; 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 ATL\textsubscript{I}\textsubscript{SAT} (ATL\textsubscript{I}SAT) for imperfect (resp. perfect) information semantics of ATL. The complexity of ATL\textsubscript{I}\textsubscript{SAT} is very high (as discussed below) and the complexity of ATL\textsubscript{I}SAT is unknown, which makes non-symbolic approaches inefficient. The complexity of ATL\textsubscript{I}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 ATL\textsubscript{I}SAT was described in (Goranko and Shkatov 2009). Subsequently, the tableau-based method was ex-
tended for checking ATL* (David 2015) and ATEL (Belardinelli 2014), an epistemic extension of ATL. The complexity (and even decidability) of ATL\textsc{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 $\Delta^P_1$-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 $\phi$, a rooted directed graph from which it is possible to extract a CGM satisfying $\phi$. 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 $\mu$-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 ATL\textsc{SAT} and ATL\textsc{ijSAT} in classes of models under given restrictions. Restrictions on the number of agents and their local states for ATL\textsc{ijSAT} result directly from the finite model property (Goranko and Drimmel 2006). The analogous result for ATL\textsc{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 $\varphi$ 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 $\varphi$. 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 $\varphi$, 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 $\varphi$ in some model of this subclass, can be determined. If the fixed initial state $i$ does not belong to this set, then there is no model of $\varphi$ 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 ATL\textsc{ij} (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 ATL\textsc{ij} (imperfect in-

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 (ATL), or is higher if the model-checking procedure is more expensive (ATL).

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 focus 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 A = {1, . . . , n}2, where each agent i ∈ A is associated with a 5-tuple AGi = (Li, ti, Acti, Pi, Ti) including:

• a finite set of local states Li = {l1, l2, . . . , ln}\n• an initial local state ti ∈ Li;\n• a finite set of local actions Acti = {a1, a2, . . . , am};\n• a local protocol Pi : L0 → 2Acti selecting the actions available at local state, where t0, . . . , ln ∈ Li, Pi(t0) \neq Ø;\n• a (partial) local transition function Ti : Li × Acti → Li, such that Ti(li, a) is defined if a ∈ Pi(li) and Ti(li, ei) = li whenever ei ∈ Pi(li) for each li ∈ Li.

We consider synchronous multi-agent systems, where each global action is a n-tuple (a1, . . . , an) with ai ∈ Acti, 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 = Act1 × · · · × Actn be the set of all global actions, \( \forall V \) 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 = (St, t, V , Act) with:

• the set St = L0 × · · · × Ln of the global states,\n• an initial state t = (t1, . . . , tn) ∈ St,\n• the global transition function T : St ∗ Act → St, such that T(s1, a) = s2 iff Titi(s1, a) = s2 for all i ∈ A.

where for global state s = (l1, . . . , ln) we denote the local component of agent i by s1 = li and for a global action a = (a1, . . . , an), ai is the local action of agent i.

We say that action a ∈ Act is enabled at s ∈ St if T(s, a) = s′ for some s′ ∈ St. We assume that at each s ∈ St there exists at least one enabled action, i.e., for all s ∈ St exist a ∈ Act, s′ ∈ St, such that T(s, a) = s′. An infinite sequence of global states and actions \( \pi = s_0a_0s_1a_1s_2 \ldots \) is called a path if T(s1, a1) = s1+1 for every i ≥ 0. Let Act(\( \pi \)) = a0a1a2 . . . be the sequence of actions in \( \pi \), and \( \pi[i] = s_i \) be the i-th global state of \( \pi \). \( \Pi_M(s) \) denotes the set of all paths in M starting at s.

2.2 Alternating-Time Logical Temporal

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 \( \langle i \rangle \). Informally, \( \langle i \rangle \varphi \) expresses that the group of agents \( \Gamma \) has a joint strategy to enforce the temporal property \( \varphi \). 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 L of ATL is defined by the following grammar: \( \varphi ::= p | \neg \varphi | \varphi \land \varphi | \langle i \rangle X \varphi | \langle i \rangle \varphi U \varphi | \langle i \rangle \neg \varphi | \langle i \rangle \varphi G \varphi \).

Let M be a model. A strategy \( \sigma_i ∈ A \) 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 \( \sigma_i : St → Acti s.t. \sigma_i(s) ∈ Pi(s) \) for each s ∈ St. A memoryless imperfect information strategy additionally satisfies \( \sigma_i(s) = \sigma_i(s') \) whenever \( s^1 = (s')^1 \). Following (Schobbens 2004), we refer to the former as I-strategies, and to the latter 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 \( \sigma_{Γ} \) for a coalition \( Γ ⊆ A \) is a tuple of strategies, one per agent \( i ∈ Γ \). We denote the set of Γ’s joint memoryless perfect (resp. imperfect) information strategies by \( \Sigma^\Gamma \) (resp. \( \Sigma^\Gamma_i \)). Additionally, let \( \sigma_{Γ} = (σ_1, . . . , σ_n) \) be a joint strategy for \( Γ = \{i_1, . . . , i_k\} \). For each s ∈ St, we define \( \sigma_{Γ}(σ) ::= (σ_1(s), . . . , σ_k(s)) \).

Definition 4 (Outcome paths). Let \( Y ∈ \{1, 2\} \). The outcome of strategy \( σ_{Γ} ∈ \Sigma^\Gamma \) in state s ∈ St is the set outM(s, \( σ_{Γ} \)) ⊆ \( \Pi_M(s) \) s.t. \( i \) ∈ \( S_M(s, \sigma_{Γ}) \) iff \( s_0 = s \) and for all \( i ∈ V \) \( α_i = \sigma_j(π[i]) \) for \( Y = 1 \) and \( α_i = \sigma_j(π[i]) \) for \( Y = 2 \).

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 $\sigma_f$ and agents (an agent) in $A \setminus \Gamma$ execute(s) an action following their protocols.

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

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

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

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

$M, s \models_Y \langle \Gamma \rangle X \varphi \iff$ there is a strategy $\sigma_1 \in \Sigma_Y$ such that $out_M(s, \sigma_1) \neq \emptyset$ and, for each path $\pi \in out_M(s, \sigma_1)$, we have $M, \pi \models_Y X \varphi$, i.e., $M, \pi[1] \models_Y \varphi$;

$M, s \models_Y \langle \Gamma \rangle \varphi_1 U \varphi_2 \iff$ there is a strategy $\sigma_1 \in \Sigma_Y$ such that $out_M(s, \sigma_1) \neq \emptyset$ and, for each path $\pi \in out_M(s, \sigma_1)$, 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 $0 \leq j < i$;

$M, s \models_Y \langle \Gamma \rangle G \varphi \iff$ there is a strategy $\sigma_1 \in \Sigma_Y$ such that $out_M(s, \sigma_1) \neq \emptyset$ and, for each path $\pi \in out_M(s, \sigma_1)$, 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 $\varphi$ is valid in $M$ (denoted $M \models \varphi$) iff $M, i \models_Y \varphi$, i.e., $\varphi$ is true at the initial state of the model $M$ under the $Y$-strategies.

By ATL$_I$ and ATL$_I$, we denote ATL with the semantics type $Y = I$ and $Y = i$, respectively.

**The problem** we consider is ATL$_I$-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 ATL$_I$ and 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 $P : \{0,1\}^n \to \{0,1\}$. We say that $P$ is Boolean positive monotonic iff $P(s_1, \ldots, s_{i-1}, 1, s_{i+1}, \ldots, s_n) = 1$, for all $1 \leq i < n$. $P$ is called Boolean negative monotonic iff $P(s_1, \ldots, s_{i-1}, 1, 0, s_{i+1}, \ldots, s_n) = 1$, for all $1 \leq i < n$.

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

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 $1 \leq i < n$.

**Definition 5** (Boolean Monotonic Theory). A theory $T$ with a signature $\Omega = (S, S_f, S_{\sigma}, ar)$, where $S$ is a non-empty set of elements called sorts, $S_f$ is a set of function symbols, $S_{\sigma}$ 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 $\Omega$ is Boolean, and all predicates and functions in $\Omega$ are monotonic.

#### 3.2 Boolean Encoding of ATL Models

First, we make some assumptions about MAS. Given a set of agents $A = \{1, \ldots, n\}$, where each agent $i$ in $A$ has a fixed set of the local states $L_i = \{l_{i1}, \ldots, l_{in}\}$ and a fixed initial local state $i_1 \in L_i$. 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 $Act_i = \{a_{i1}', \ldots, a_{in}'\}$ and an action $a_i'$ can move the agent $i$ from any local state to local state $l_{ij}'$. 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: $T_i(l_{ij}^k, a_i^j) = l_{ij}^j$ if $a_i^j \in P_i(l_{ij}^k)$, for any $l_{ij}^k \in L_i$ and $1 \leq j \leq n_i$.

Next, we represent every single agent $i$ with a given $AG_i = (L_i, i_1, Act_i, P_i, T_i)$ 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 $(l_{ij}^k, a_i^j)$ means that the local action $a_i^j$ is not available at the local state $l_{ij}^k$, and $1$ stands for the availability. This table can be represented by a bit vector $tb_i = (lp_i[1], \ldots, lp_i[n_i])^2$, 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 $M = (\mathcal{S}t, \mathcal{I}, T, V)$ induced by a MAS is a product of $AG_i$ for $i \in A$, 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 $\mathcal{P}V$ of $|\mathcal{S}t| \times |\mathcal{P}V|$ saves which propositional variables are true in which global states. Then, let $vb = (vb_1, \ldots, vb_k)$ be a bit vector, where $k = |\mathcal{S}t| \cdot |\mathcal{P}V|$, 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 $|\mathcal{P}V|$ 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 $|\mathcal{P}V|$ of the propositional variables and a fixed number $n$ of agents with fixed numbers $|\mathcal{S}t|, |\mathcal{P}V|$, and $1 \leq j \leq n_i$. Thus, consider models that can be represented by a bit vector $v_M$ consisting of exactly $n_M = |\mathcal{S}t|^2 + \cdots + |\mathcal{P}V|^2 \cdot |\mathcal{S}t| \cdot |\mathcal{P}V|$ 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 $i = 1, \ldots, n$, $\|$
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 \( Y \in \{1, i\} \) the predicate \( \text{Model}^Y_{g, \phi}(v_M) \) is defined such that for the bit vector \( v_M \),

\[
\text{Model}^Y_{g, \phi}(v_M) = 1 \text{ if and only if } M, g \models \phi
\]

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 \( \text{Model}^Y_{g, \phi}(v_M) \) is neither positive nor negative monotonic w.r.t. \( v_M \), for \( Y \in \{1, i\} \).

**Proof.** It follows from a similar result for CTL (Klenze, Bayless, and Hu 2016) as \( \text{ATL}_4(\text{ATL}_4) \) subsumes CTL. \( \square \)

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 \( M\text{App}^Y_{g, \phi}(V_{m_1}, V_{m_2}) \), for \( Y \in \{1, i\} \), defined over two vectors, is introduced (see Sec. 5.1). This predicate is positive monotonic w.r.t. \( V_{m_1} \), and negative monotonic w.r.t. \( V_{m_2} \), and has the property that for every vector \( v_M \), \( M\text{App}^Y_{g, \phi}(V_{m_1}, v_M) = \text{Model}^Y_{g, \phi}(v_M) \). To compute \( M\text{App}^Y_{g, \phi}(v_1, v_2) \) for vectors \( v_1, v_2 \), s.t. \( v_2[j] \leq v_1[j] \) for every \( j \), the algorithm \( S\text{App}^Y \) is used (see Sec. 4.2). It takes as an input a formula \( \phi \) and a partial model \( M_{par} \) (see Sec. 4.1), and determines an approximation of a set of states satisfying \( \phi \) in a model \( M \) which extends \( M_{par} \). This means that if a state \( \iota \) does not belong to the approximation, then it cannot satisfy \( \phi \) 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 \( S\text{App}^Y \) 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 formulae 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: \( \text{solve} \) and \( MC \), which are defined and proved in the next subsection. The proofs are conducted for the ATL formulae under the I semantics. The proofs for the i semantics are omitted since they are exactly the same.

### 3.4 Functions \( \text{solve} \) and \( MC \)

In order to compute the value of the predicate \( \text{Model}^Y_{g, \phi}(v_M) \) for a given \( M \) and \( Y \in \{1, i\} \), we define a new function, called \( \text{solve}^Y_{g, \phi}(v_M) \). This function returns a set of states of \( M \) such that

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

and preserves the following monotonicity properties.

**Theorem 2.** The function \( \text{solve}^Y_{g, \phi}(v_M) \), for \( Y \in \{1, i\} \) is

1. positive monotonic w.r.t. \( V_B \) for \( \phi \in \{p, p \land q, \langle \Gamma \rangle X p, \langle \Gamma \rangle G p, \langle \Gamma \rangle p U q\} \).
2. negative monotonic w.r.t. \( V_B \) for \( \phi = \neg p \).
3. positive and negative monotonic w.r.t. \( TB_i \) for \( i \in A \) if \( \phi \in \{p, \neg p, p \land q\} \).
4. positive monotonic w.r.t. \( TB_i \) for each \( i \in \Gamma \) if \( \phi \in \{\langle \Gamma \rangle X p, \langle \Gamma \rangle G p, \langle \Gamma \rangle p U q\} \).
5. negative monotonic w.r.t. \( TB_i \) for \( i \in A \setminus \Gamma \) if \( \phi \in \{\langle \Gamma \rangle X p, \langle \Gamma \rangle G p, \langle \Gamma \rangle p U q\} \), where \( p, q \in \mathcal{P} V \).

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 \( p \in \mathcal{P} V \) holds does not affect pre-existing strategies involving \( p \). Moreover, deleting a local transition for agent \( B \in \Gamma \) can reduce only a number of outcome paths of a strategy \( \sigma \). 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 \( \text{solve}^Y_{g, \phi}(v_M) \) for a formula \( \phi \) and \( Y \in \{1, i\} \), an evaluation function \( \text{MC}^Y \) is defined for an unary operator \( op \) and \( \text{MC}^Y \) 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 \( \phi = p \in \mathcal{P} V \), then for a given model \( M \), \( \text{solve}^Y_{g, \phi}(v_M) \) returns the set of states of \( M \) in which \( p \) holds. Otherwise, \( \text{solve}^Y_{g, \phi}(v_M) \) takes the top-most operator \( op \) of \( \phi \) and solves its argument(s) recursively using the function \( \text{MC}^Y \) and applies \( \text{MC}^Y \) or \( \text{MC}^Y \) to the returned set(s) of states. Notice that \( \text{MC}^Y(\lnot, Z, v_M) \) computes the compliment of \( Z \) in \( M \), i.e. \( St \setminus Z \). Similarly, \( \text{MC}^Y(\land, V_1, V_2) \) computes the intersection of \( V_1 \) and \( V_2 \) in \( M \). If \( op \in \{\langle \Gamma \rangle X, \langle \Gamma \rangle G, \langle \Gamma \rangle U\} \) 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 \( \text{MC}^Y \) for an unary operator \( op \) and \( \text{MC}^Y \) for a binary operator \( op \), for \( Y \in \{1, i\} \) are:

1. positive monotonic w.r.t. \( V_B \) for \( op \in \{\land, \langle \Gamma \rangle X, \langle \Gamma \rangle G, \langle \Gamma \rangle U\} \). 2. negative monotonic w.r.t. \( V_B \) for \( op = \lnot \). 3. positive and negative monotonic w.r.t. \( TB_i \) for \( i \in A \) and \( op \in \{\land, \lnot\} \). 4. positive monotonic w.r.t. \( TB_i \) for \( i \in \Gamma \) and \( op \in \{\langle \Gamma \rangle X, \langle \Gamma \rangle G, \langle \Gamma \rangle U\} \). 5. negative monotonic w.r.t. \( TB_i \) for \( i \in A \setminus \Gamma \) and \( op \in \{\langle \Gamma \rangle X, \langle \Gamma \rangle G, \langle \Gamma \rangle U\} \).

4 Approximating ATL Models

This section shows how to approximate models for ATL using SMT. 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 overapproximation of the set of all states satisfying the given formula in some model of the class.

### 4.1 Construction of $M_{\text{over}}$ and $M_{\text{under}}$

Given a set of agents $A = \{1, \ldots, n\}$, following Def. 1 we define a partial protocol function: $CP_i : L_i \times Act_i \rightarrow \{0, 1, undef\}$. 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 $CV : St \times PV \rightarrow \{0, 1, undef\}$ is called a partial model, denoted by $M_{\text{par}}$. 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 $M_{\text{par}}$, total models $M^r_{\text{under}}$ and $M^f_{\text{over}}$, for $\Gamma \subseteq A$, are constructed. First, for every agent $i \in A$ we define a necessary local protocol $P_i = L_i \rightarrow 2^{Act_i}$ and a possible local protocol $\overline{P}_i = L_i \rightarrow 2^{Act_i}$, where:

1. if $CP_i(l_i, a_i) = 1$ then $a_i \in P_i(l_i)$ and $a_i \not\in \overline{P}_i(l_i)$,
2. if $CP_i(l_i, a_i) = 0$ then $a_i \not\in P_i(l_i)$ and $a_i \not\in \overline{P}_i(l_i)$,
3. if $CP_i(l_i, a_i) = undef$ then $a_i \not\in P_i(l_i)$ and $a_i \not\in \overline{P}_i(l_i)$.

Notice that the possible local protocol is an extension of the necessary local protocol, i.e., for every local state $l_i$, $P_i(l_i) \subseteq \overline{P}_i(l_i)$. Similarly, total valuations of the propositional variables are defined: a necessary valuation $V : St \rightarrow 2^{PV}$ and a possible valuation $\overline{V} : St \rightarrow 2^{PV}$ such that:

1. if $CV(g, p) = 1$ then $p \in V(g)$ and $p \not\in \overline{V}(g)$,
2. if $CV(g, p) = 0$ then $p \not\in V(g)$ and $p \not\in \overline{V}(g)$,
3. if $CV(g, p) = undef$ then $p \not\in V(g)$ and $p \not\in \overline{V}(g)$.

Thus, for every global state $g \in St$ we have $V(g) \subseteq \overline{V}(g)$.

The total model $M^r_{\text{under}}$ is defined as in Def. 2 of all agents $i \in \Gamma$ with $AG_i = (L_i, t_i, Act_i, P_i, T_i)$, agents $j \in A \setminus \Gamma$ with $AG_j = (L_j, S_j, Act_j, T_j)$, and for the valuation of the propositional variables $V$. The total model $M^f_{\text{over}}$ is defined as in Def. 2 of all agents $i \in \Gamma$ with $AG_i = (L_i, t_i, Act_i, \overline{P}_i, T_i)$, agents $j \in A \setminus \Gamma$ with $AG_j = (L_j, S_j, Act_j, \overline{T}_j)$, and for the valuation of the propositional variables $\overline{V}$.

### Definition 7

Let $M = (St, t, T, CV)$ and $M' = (St, t, T, CV')$ be (partial) models defined over $PV$ and induced by agents $A$ with $AG_i = (L_i, t_i, Act_i, CP_i, T_i)$ and $AG'_i = (L_i, t_i, Act_i, CP'_i, T_i)$ for $i \in A$, 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 $CP_i(l_i, a_i) = 0$,
- (3) if $CV'(g, p) = 1$ then $CV(g, p) = 1$,
- (4) if $CV'(g, p) = 0$ then $CV(g, p) = 0$.

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

### Theorem 4

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

- $v_{M^r_{\text{under}}}[TB_i] \leq v_{M^f_{\text{under}}}[TB_i] \leq v_{M^f_{\text{over}}}[TB_i]$ for $i \in \Gamma$, and
- $v_{M^f_{\text{under}}}[TB_i] \leq v_{M^f_{\text{over}}}[TB_i]$ for $i \in A \setminus \Gamma$, and
- $v_{M^f_{\text{under}}}[VB] \leq v_{M^f_{\text{over}}}[VB]$.

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

### Proof

Follows from the construction of over and under models.

### 4.2 Algorithm $SA_{App}$

The heart of the approach is Algorithm 1. For a given strategy type $Y \in \{1, 1\}$, it has on input: an ATLf formula $\phi$, a (partial) model $M_{\text{par}}$, and $\lambda$ parameter, which can have one of two values over or under. The algorithm returns a set of states, which for $\lambda = \text{under}$ specifies under-approximation of a set of states satisfying $\phi$ and for $\lambda = \text{over}$ determines the over-approximation of this set. If $\phi = \rho$ is a propositional variable then $SA_{App}^r(\rho, M, \lambda)$ computes a set of states satisfying $\rho$ in the total model $M^r$. If $\phi = \neg \varphi$, then $SA_{App}^r(\phi, M, \lambda)$ computes the approximation for a subformula $\varphi$ performing $SA_{App}(\varphi, M, \lambda)$, where $\lambda = \text{under}$ if $\lambda = \text{over}$, and otherwise. Next, the function $MC^r$ determines the compliment of the set $SA_{App}^r(\varphi, M, \lambda)$ in $M^r$. If $\phi = (\langle x \rangle \chi \varphi)$, then $SA_{App}^r(\phi, M, \lambda)$ computes the approximation for a subformula $\varphi$ performing $Z = SA_{App}(\varphi, M, \lambda)$ and the function $MC^r$ determines the set of states satisfying $\langle x \rangle \chi \varphi$ in $M^r$, where $\chi$ is a propositional formula satisfied only by states belonging to $Z$. In fact, $MC^r$ 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 $M_{\text{par}}$ and $M'_{\text{par}}$ be two (partial) models s.t. $M_{\text{par}}$ is compatible with $M'_{\text{par}}$. Then, for $Y \in \{1, 1\}$, $SA_{App}^Y(\phi, M_{\text{par}}, \overline{over}) \subseteq SA_{App}^Y(\phi, M'_{\text{par}}, \overline{over})$ and $SA_{App}^Y(\phi, M_{\text{par}}, \overline{under}) \subseteq SA_{App}^Y(\phi, M'_{\text{par}}, \overline{under})$.

### Proof

By a structural induction on a formula $\phi$. It is the same for both semantics: $Y = 1$ and $Y = 1$. We show the first case. Since $M_{\text{par}}$ is compatible with $M'_{\text{par}}$ then $v_{M_{\text{par}}}[\phi][j] \leq v_{M'_{\text{par}}}[\phi][j]$ for any $\Gamma$ and $j \in \{VB, TB_i\}$ with $i \in \Gamma$ and $v_{M_{\text{par}}}[\phi][j]$ greater for any $\Gamma$ and $j = TB_i$. We show $\phi = p \in PV$, then from the definition of the algorithm, $SA_{App}(p, M_{\text{par}}, \overline{over})$ returns the set of the states satisfying $p$ in $M_{\text{over}}$ and $SA_{App}(p, (M'_{\text{par}}, \overline{over})$...
Algorithm 1 Algorithm SApp

Input: φ, M_{par}, λ
Output: a set of states
1: if φ ∈ \mathcal{P}(\mathcal{W}) then
2: return \{g ∈ St : M^A, g \models_\mathcal{V} \phi\}
3: else if φ = op(ψ) then
4: if op is ¬ then // negative monotonic
5: if λ = over then
6: Z := SApp^{\mathcal{V}}(\psi, M_{par}, under)
7: return MC^\mathcal{V}(op, Z, v_{M^A_{\text{under}}})
8: else
9: Z := SApp^\mathcal{V}(\psi, M_{par}, over)
10: return MC^\mathcal{V}(op, Z, v_{M^A_{\text{under}}})
11: else if φ = op(ψ_1, ψ_2) for op ∈ \{\langle X \rangle, \langle Y \rangle \} then
12: Z_1 := SApp^\mathcal{V}(\psi_1, M_{par}, λ)
13: Z_2 := SApp^\mathcal{V}(\psi_2, M_{par}, λ)
14: if op is \langle U \rangle then
15: return MC^\mathcal{V}(op, Z_1, Z_2, v_{M^I_{\text{under}}})
16: else if op is ∧ then
17: return MC^\mathcal{V}(op, Z_1, Z_2, v_{M^I_{\text{under}}})

returns the set of the states satisfying \( p \) in \( (M')^\mathcal{A}_{\text{over}} \). Thus SApp^\mathcal{V}(φ, M_{par}, over) ⊆ SApp^\mathcal{V}(φ, M_{par}', over) since \( v_{M^I_{\text{over}}} \mid V\mathcal{B} \) ⊆ \( v_{(M')^\mathcal{A}_{\text{over}}} \mid V\mathcal{B} \). Similarly, SApp^\mathcal{V}(φ, M_{par}', under) ⊆ SApp^\mathcal{V}(φ, M_{par}, under) since \( v_{(M')^\mathcal{A}_{\text{under}}} \mid V\mathcal{B} \) ≤ \( v_{M^I_{\text{under}}} \mid V\mathcal{B} \).

The induction step. We show the proof for the unary operators ¬, \langle X \rangle, \langle Y \rangle. The proofs for the binary operators Until and ∧ are similar. Induction assumption (IA): the thesis holds for a formula \( ψ \). Induction hypothesis (IH): the thesis holds for φ = op(ψ).

If φ = ¬ψ.

Let Z = SApp^\mathcal{V}(ψ, M_{par}, under) and Z' = SApp^\mathcal{V}(ψ, M_{par}', under), then Z' ⊆ Z from IA. Then, MC^\mathcal{V}(¬, Z, v_{M^A_{\text{under}}}) ⊆ MC^\mathcal{V}(¬, Z', v_{M^A_{\text{under}}}) since MC^\mathcal{V} for op = ¬ returns the compliment of Z and Z', respectively. Next, MC^\mathcal{V}(¬, Z', v_{M^A_{\text{under}}}) ⊆ MC^\mathcal{V}(¬, Z', v_{M^A_{\text{under}}}) since MC\mathcal{V} is negative monotonic w.r.t. V\mathcal{B} and \( T\mathcal{B}_i \) for \( i ∈ \mathcal{A} \) from Theorem 3. Thus, MC^\mathcal{V}(¬, Z, v_{M^I_{\text{under}}}) ⊆ MC^\mathcal{V}(¬, Z', v_{M^I_{\text{under}}}) and SApp^\mathcal{V}(φ, M_{par}, over) ⊆ SApp^\mathcal{V}(φ, M_{par}', over).

Let Z = SApp^\mathcal{V}(ψ, M_{par}, over) and Z' = SApp^\mathcal{V}(ψ, M_{par}', over), then Z' ⊆ Z from IA. Then, MC^\mathcal{V}(¬, Z', v_{M^I_{\text{over}}}) ⊆ MC^\mathcal{V}(¬, Z, v_{M^I_{\text{over}}}) since MC\mathcal{V} for op = ¬ returns the compliment of Z' and Z, respectively. Next, MC\mathcal{V}(¬, Z, v_{M^I_{\text{over}}}) ⊆ MC\mathcal{V}(¬, Z, v_{(M')^\mathcal{A}_{\text{over}}}) since MC\mathcal{V} is negative monotonic w.r.t. V\mathcal{B} and \( T\mathcal{B}_i \) for \( i ∈ \mathcal{A} \) from Theorem 3. Thus, MC\mathcal{V}(¬, Z, v_{(M')^\mathcal{A}_{\text{over}}}) ⊆ MC\mathcal{V}(¬, Z, v_{M^I_{\text{over}}}) and SApp\mathcal{V}(φ, M_{par}', under) ⊆ SApp\mathcal{V}(φ, M_{par}, under).

The algorithm SApp^\mathcal{V}, for a total model M compatible with a partial model M_{par}, computes over and under-approximation of solve^\mathcal{V}(v_M), for \( Y ∈ \{I, i\} \). More precisely, SApp^\mathcal{V}(φ, M_{par}, over) returns a set of states, which is an over-approximation of solve^\mathcal{V}(v_M). This means that if \( t ∈ solve^\mathcal{V}(v_M) \), then \( t ∈ SApp^\mathcal{V}(φ, M_{par}, over) \). Clearly, if \( t ∉ SApp^\mathcal{V}(φ, M_{par}, over) \), then there is no model M extending M_{par} such that M, t \models_\mathcal{V} φ. Similarly, SApp^\mathcal{V}(φ, M_{par}, under) computes an under-approximation of solve^\mathcal{V}(v_M). This means that if \( t ∈ SApp^\mathcal{V}(φ, M_{par}, under) \) then \( t ∈ solve^\mathcal{V}(v_M) \).

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^\mathcal{V}(φ, M_{par}, under) ⊆ solve^\mathcal{V}(v_M) ⊆ SApp^\mathcal{V}(φ, M_{par}, over), for \( Y ∈ \{I, i\} \).

Proof. Follows from Theorem 5 and the fact that SApp^\mathcal{V}(φ, M, under) = SApp^\mathcal{V}(φ, M, over) = solve^\mathcal{V}(v_M), for \( Y ∈ \{I, i\} \), since M is total.

5 Satisfiability Procedure

Basing on the SMT 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\mathcal{V} algorithm as the theory solver for ATL.
5.1 Monotonic Predicate $M_{App}$

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

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

5.2 Satisfiability Tool

Below we describe how the algorithm searching for values satisfying $M_{App}^Y_{g,o}(V_{m_1}, V_{m_2})$, on the basis of which the tool solving ATL+SAT with restrictions, is built. For a fixed $Y \in \{I, I\}$ the following input and output are defined.

**Input:** (a) an ATL formula $\phi$, (b) model requirements determining a partial model $M_{par}$.

**Output:** a model satisfying $\phi$, 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 $M_{App}^Y_{g,o}(V_{m_1}, V_{m_2})$ is defined, and $asq(i)$ 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 $S_{App}^Y(\phi, M_{par, over})$.
3. If $\phi \in S_{App}^Y(\phi, M_{par, over})$, then (a) if all variables of $V_m$ have assigned values and $v$ is the determined vector of Boolean values, then $M_{App}^Y_{g,o}(v, v) = 1$. Return the model represented by $v$. (b) otherwise: $d := d + 1$, and the SAT-solver assigns a value to the variable $asq(d)$ 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 $\phi \notin S_{App}^Y(\phi, M_{par, over})$, 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 $asq(c)$, and go to step 2. (b) if $d = 0$ there is no model meeting the requirements and satisfying $\phi$. Return UNSAT.

**Example 1.** Let us consider the formula $\beta = \neg p \wedge [\langle 0 \rangle X p]$. We want to check if (under perfect information semantics) there is a model for $\beta$ 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 ($V_B$).

Initially, the SAT solver’s assignment to variables of $V_B$ 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 is shown as a matrix. The values at $(s, t)$ determine if the transition from state $s$ to $t$ is enabled ($T \text{ true }$, solid line), disabled ($F \text{ false }$, nothing), or enabled in overapproximation and disabled in underapproximation (U undefined, dashed line). The colours of global states represent the valuation of $V_B$ variables: green - the property $p$ holds, black - does not hold, and yellow - holds in $v_{M_{under, over}}$, and does not hold in $v_{M_{under}}$. The double circles denote the initial states.

We start with computing $S_{App}^I(\beta, M_{par, over})$.

According to Algorithm 1 (line 14) we have:

$$Z_1 = S_{App}^I(\neg p, M_{par, over}) = MC^I(\neg, S_{App}(p, M_{par, under}), v_{M_{under}}) = MC^I(\neg, 0, v_{M_{under}}) = \{0, 1, 2, 3\},$$

and

$$Z_2 = S_{App}^I(\langle 0 \rangle X p, M_{par, over}) = MC^I(\langle 0 \rangle X, S_{App}(p, M_{par, over}), v_{M_{over}}) = MC^I(\langle 0 \rangle X, v_{M_{over}}) = \{0, 1, 2, 3\}^4.$$

Finally, we have the set: $MC^I(\land, Z_1, Z_2, v_{M_{over}}) = Z_1 \cap Z_2 = \{0, 1, 2, 3\}$. 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 $S_{App}^I(\beta, M_{par, over})$.

Assume, that after several assignments our partial model $M_{par}$ is as in Fig. 2. All variables of TB are assigned, and only one of variables of TB0 and one of V B are undefined. As before, the algorithm computes $S_{App}^I(\beta, M_{par, over})$:

$$Z_1 = S_{App}^I(\neg p, M_{par, over}) = MC^I(\neg, S_{App}(p, M_{par, under}), v_{M_{under}}) = MC^I(\neg, 0, v_{M_{under}}) = \{1, 2, 3\},$$

and

$$Z_2 = S_{App}^I(\langle 0 \rangle X p, M_{par, over}) = MC^I(\langle 0 \rangle X, S_{App}(p, M_{par, over}), v_{M_{over}}) = MC^I(\langle 0 \rangle X, v_{M_{over}}) = \{0, 2, 3\}.$$

Then we have: $MC^I(\land, Z_1, Z_2, v_{M_{over}}) = Z_1 \cap Z_2 = \{2\}$. The initial state 0 does not belong to the set, so it is not possible to extend $M_{par}$ to a model for $\beta$. 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 all.
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/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 reduction strategies 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 TATL – large formulae satisfied by relative small models.

Table 1 presents experimental results for perfect information.

Table 2 presents experimental results for randomly generated formulae of ATL_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 \( \phi \equiv (\neg \text{next} \land \langle 1 \rangle F \neg \text{next} \land \langle 0 \rangle G(\text{next} \rightarrow \langle 1 \rangle F \text{win}) \rightarrow \langle 1 \rangle F \text{win}) \). Intuitively, \( \phi \) 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. Formulate like \( \phi \) are known to be valid for ATL_i but not for ATL (Bulling and Jamroga 2014). Our tool, using two local states per agent, determined the negation of \( \phi \) to be satisfiable for ATL_i (80 sec.) and unsatisfiable for ATL (11 sec.), which demonstrates that both functionalities of MsATL are important.

7 Conclusions

The paper introduced a new method exploiting SMT 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.

---

Table 1: Experimental results for perfect information.

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

Table 2: Experimental results for imperfect information.

---

---

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

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


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


