Satisfiability Checking of Strategy Logic with Simple Goals

Magdalena Kacprzak\textsuperscript{1}, Artur Niewiadomski\textsuperscript{2}, Wojciech Penczek\textsuperscript{3}

\textsuperscript{1}Bialystok University of Technology, Bialystok, Poland
\textsuperscript{2}Siedlce University, Faculty of Exact and Natural Sciences, Siedlce, Poland
\textsuperscript{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

In this paper, we introduce a new method of the satisfiability (SAT) checking for Simple-Goal Strategy Logic ($\text{SL}[\text{SG}]$), using symbolic Boolean model encoding and the SAT Modulo Monotonic Theories techniques, which was implemented into the tool SGSAT. To the best of our knowledge, this is the only tool solving the SAT problem for $\text{SL}[\text{SG}]$. Its applications include process synthesis, developing controllers as well as automatic planners in multi-agent scenarios.

1 Introduction

An important area of research in artificial intelligence systems is the development of techniques for reasoning on strategies, especially in open-systems, where the interaction between the system and the environment is viewed as a strategy game. Recently, a lot of work in the field of verification of multi-agent system (MAS) has been devoted to establishing what strategic properties in the system agents have (Jamroga and van der Hoek 2004; Ágotnes, Goranko, and Jamroga 2007; Walther, van der Hoek, and Wooldridge 2007; Pauly 2002; Kacprzak and Penczek 2005). The two main lines of research involve specifications in the formalisms of Alternating-time Temporal Logic (ATL and ATL*) (Alur, Henzinger, and Kupferman 2002) supported by the tools like MOCHA (Alur et al. 2001) or MCMAS (Lomuscio and Raimondi 2006) and its generalization, Strategy Logic (SL) (Mogavero, Murano, and Vardi 2010) supported by the tools like MCMAS-SLK (Cermák et al. 2014).

SL was originally defined over perfect recall and complete information semantics. It extends Linear-time Temporal Logic (LTL) (Pnueli 1977) by strategy quantifiers and agent bindings. The syntactic separation of these concepts makes it possible to construct complex formulae in which different temporal goals referring to the same strategies can be combined and nested, as well as different players can share the same strategy. Strategies are treated as a first order concept over strategy and agent variables, which allows for expressing interesting properties like strategic plans over temporal goals, Nash equilibria or Stackelberg equilibria.

High expressivity of SL has its price, namely the high computational complexity. The model checking problem (MC) for SL is non-elementarily decidable - $k$-$\text{EXPSPACE}$-hard in the alternation number $k$ of quantifications in the specification (Mogavero et al. 2014). The satisfiability problem (SAT) for SL is highly undecidable - $\Sigma_1^1$-hard. Moreover, the logic does not have the bounded-tree model property (Mogavero et al. 2017). Since this makes SL quite limited in its practical application, restrictions of SL are of interest. These include One-Goal Strategy Logic ($\text{SL}[1\text{G}]$) having the SAT problem in $\text{ZEXPTIME}$ (Mogavero et al. 2014) and its sublogic Simple-Goal Strategy Logic ($\text{SL}[\text{SG}]$) for which MC is $\text{P}$-complete (Belardinelli et al. 2019).

This paper offers a new method of SAT checking for $\text{SL}[\text{SG}]$, extending the one for ATL (Kacprzak, Niewiadomski, and Penczek 2020) and uses symbolic Boolean model encoding and the SAT Modulo Monotonic Theories (SMMT) techniques (Bayless et al. 2015). The method was implemented into the tool SGSAT, which, to our best knowledge, is the only one solving the SAT problem for $\text{SL}[\text{SG}]$. Applications of $\text{SL}[\text{SG}]$ include specification of Stackelberg equilibria, coercion-resistance, and module checking for strategic abilities (Belardinelli et al. 2019; Tabatabaei, Jamroga, and Ryan 2016; Jamroga and Murano 2014).

Typically, given a SAT method for a logic, one can solve the MC problem using the automata approach. In this paper we show that it is possible to solve the SAT problem, given a MC method. We develop an efficient method for generating candidate models for an $\text{SL}[\text{SG}]$ formula, and then use an existing MC method for checking whether the formula holds in one of these models. The best analogy to our approach is how SAT-solvers check satisfiability of a propositional formula. Clearly, only some candidate valuations are generated thanks to sophisticated SAT-algorithms like CDCL (Silva and Sakallah 2003). Similarly, in our case, only some candidate models are generated thanks to the algorithms for SMMT. This is where monotonicity of our theory is exploited, which allows for a substantial reduction in the number of generated model candidates. Therefore, our SAT method for $\text{SL}[\text{SG}]$ at the stage of finding candidate models can be as efficient as a SAT-solver is in finding candidate valuations for a propositional formula.

Our paper is not the first one adopting that approach. Here we build on the results for CTL (Klenze, Bayless, and Hu 2016) and ATL (Niewiadomski et al. 2020). The main contributions for $\text{SL}[\text{SG}]$ consist in: a new encoding method for $\text{SL}[\text{SG}]$ models, new proofs of monotonicity of the relation $\models$ for $\text{SL}[\text{SG}]$ in some cases, new definitions of over-
and under-approximations for SL[SG] partial models, new proofs of the correctness of the method for the formulae prefixed with strategy quantifiers and bindings, and a new implementation and the tool.

It is worth mentioning that with some technical difficulties our approach for SL[SG] can be extended to SL[1G] and SL.

Outline. The related work is discussed in Sec. 2. Next, the necessary definitions are given in Sec. 3. In Sec. 4 the Boolean encoding of MAS and its model is introduced. The proof of the monotonicity property for SL[SG] is in Sec. 5, followed by the construction of the monotonic approximation for a given class of models, in Sec. 6. Sec. 7 and 8 describe the satisfiability procedure, its implementation, experimental results, and provide conclusions.

2 Related Work

To overcome the problem of high computational complexity, several restrictions of SL are considered. The following logics have been defined over the original semantics: Nested-Goal Strategy Logic (SL[NG]), Boolean Goal Strategy Logic (SL[BG]), Conjunctive-Goal Strategy Logic (SL[CG]), Disjunctive-Goal Strategy Logic (SL[DG]), and the already mentioned One-Goal Strategy Logic (SL[1G]) (Mogavero et al. 2014; Mogavero, Murano, and Sauro 2013). The SAT problem for SL[NG] and SL[BG] is undecidable and strategies are non-behavioural. SAT for SL[CG] and SL[DG] is an open problem while the strategies are behavioural. SL[1G], unlike the previous logics, enjoys both desired properties: the decidable SAT problem and behavioural strategies which is fundamental for proving decidability (Mogavero et al. 2014). The SAT procedure for SL[1G] is given in (Mogavero et al. 2017).

In (Acar, Benerecetti, and Mogavero 2019) a fragment of SL is considered, which prevents strategic quantification within the scope of temporal operators, called Flat Conjunctive-Goal Strategy Logic (FSL[CG]). This logic still allows for expressing the existence of mutual intention and equilibrum, but its SAT problem is PSPACE-complete while the MC problem is 2EXPTIME-HARD.

A model checker for Strategy Logic with Knowledge for systems with memoryless strategies and incomplete information is introduced in (Cermák et al. 2018). This tool is an extension of the model checker MCMAS, which has also been enhanced with verification of SL[1G] defined for systems with perfect recall and complete information (Cermák, Lamuscio, and Murano 2015). An epistemic extension of SL with incomplete information and uniform and coherent strategies is also defined in (Belardinelli et al. 2017).

3 Strategy Logic with Simple Goals

We deal with a fragment of SL, restricted to “simple” goals (Belardinelli et al. 2019). However, the semantics is defined over interpreted systems (IS) (Fagin et al. 1995; Lamuscio and Raimondi 2006) instead of concurrent game structures.

3.1 Multi-agent System and Interpreted System

We start with a definition of multi-agent systems.

Definition 1 (MAS). A multi-agent system (MAS) consists of n agents \( A = \{1, ..., n\} \), where each agent \( i \in A \) is associated with a 7-tuple \( A_i = \{L_i, t_i, \Delta c_i, P_i, T_i, \Phi V_i, V_i\} \) including:

1. a finite non-empty set of local states \( L_i = \{l_i^1, l_i^2, ..., l_i^n_i\} \);
2. an initial local state \( t_i \in L_i \);
3. a finite non-empty set of local actions \( \Delta c_i = \{a_i^1, a_i^2, ..., a_i^m_i\} \);
4. a local protocol \( P_i : L_i \rightarrow 2^{\Delta c_i} \setminus \{\emptyset\} \) selecting the actions available at each local state;
5. a (partial) local transition function \( T_i : L_i \times \Delta c_i \rightarrow L_i \) such that \( T_i(t_i, \alpha) \) is defined iff \( \alpha \in P_i(t_i) \), where \( \Delta c_i = \bigcap_{i \in A} \) \( \Delta c_i \), is the set of joint (global) actions of all agents; \( \alpha^j \) is the action of agent \( i \in A \) in the joint action \( \alpha \in \Delta c_i \); (6) a finite non-empty set of local propositions \( \Phi V_i = \{p_i^1, p_i^2, ..., p_i^n_i\} \); (7) a local valuation function \( V_i : L_i \rightarrow 2^{\Phi V_i} \).

For every non-empty set \( \Gamma \subseteq A \), the set of shared actions of agents \( \Gamma \) is determined \( \Delta c_{\Gamma} = \bigcap_{i \in \Gamma} \Delta c_i \). Such a set must be non-empty if we want to ensure the existence of a non-empty set of shared strategies for \( \Gamma \) (i.e. a set of strategies that can be used by every \( i \in \Gamma \)).

Notice that we consider synchronous multi-agent systems, where each global action is a n-tuple \( (a_1, ..., a_n) \) with \( a_i \in \Delta c_i \), i.e., each agent performs one local action. To describe the behavior of a MAS, its model is defined.

Definition 2 (Model). Let \( PV = \bigcup_{i=1}^{n} \Phi V_i \) be the union of the local propositions. The model for MAS is a 4-tuple \( M = (S, \iota, T, V) \), where

1. \( S = L_1 \times \cdots \times L_n \) is a set of the global states,
2. \( \iota = (t_1, ..., t_n) \in S \) is the initial global state,
3. \( T : S \times \Delta c_i \rightarrow S \) is the partial global transition function, such that \( T(g, \alpha) = g' \) if \( T_i(g_i, \alpha) = g' \) for all \( i \in A \), where for a global state \( g = (l_1, ..., l_n) \), by \( g_i = l_i \) we denote the local component of agent \( i \),
4. \( V : S \rightarrow 2^{\Phi V} \) is the valuation function such that \( V((l_1, ..., l_n)) = \bigcup_{i=1}^{n} V_i(l_i) \).

We say that action \( \alpha \in \Delta c_i \) is enabled at \( g \in S \) if \( T(g, \alpha) = g' \) for some \( g' \in S \). We assume that at each \( g \in S \) there exists at least one enabled action, i.e., for each \( g \in S \) there are \( \alpha \in \Delta c_i \) and \( g' \in S \) s.t. \( T(g, \alpha) = g' \). A path is a (finite or infinite) sequence of global states \( \pi \in S^* \cup S^\omega \) such that for each \( j \geq 1, \pi_{j+1} = T(\pi_j, \alpha_j) \) for some joint action \( \pi_j \in \Delta c \), where \( \pi_j \) denotes the j-th state of \( \pi \). We distinguish between finite paths, or histories of \( S^* \), and infinite paths, or computations of \( S^\omega \). For a path \( \pi \) and \( j \geq 1, \pi_{j} \) denotes the initial history of length \( j \), and \( last(\pi) \) is the last element in history \( \pi \).

Example 1. Consider the following MAS:

- \( A = \{1, 2\} \), \( L_1 = \{1, 2, 3\}, t_1 = 1 \), \( L_2 = \{1, 2\}, t_2 = 1, \)
- \( \Delta c_1 = \{1, 2\}, P_1 = \{(1, 1), (2, 1), (3, 1), (2, 2)\}, \)
- \( \Delta c_2 = \{1, 2\}, P_2 = \{(1, 1), (2, 2)\}, \)
- \( T_1 = \{(1, 1, 1, 2), (1, 1, 2, 3), (2, 1, 1, 3), (2, 1, 2, 3), (3, 1, 1, 2), (3, 1, 2, 2), (3, 2, 2, 2), (3, 2, 1, 1)\}, \)
- \( T_2 = \{(1, 1, 1, 1), (1, 2, 1, 1), (1, 1, 2, 2), (1, 2, 2, 2),\}

\( ^4 \)The environment component may be added here with no technical difficulty.
(2, (1, 2), 2), (2, (2, 2), 2)); \( PV_1 = \{p\}, PV_2 = \{q\} \), 
\( V_1 = \{(1, \emptyset), (2, \{p\}), (3, \emptyset)\}, V_2 = \{(1, \emptyset), (2, \{q\})\} \),
depicted in Fig. 1 (left), while at the right hand side of the figure its induced model is presented. For a better read-
ability each joint action is represented by the integer cor-
responding to the action’s position sorted lexicographically. 
That is, 1 states for (1, 1), 2 represents (1, 2), and so on.

3.2 Syntax and Semantics of SL[SG]

The following definitions, needed for defining the syntax of 
SL[SG], are adopted from (Belardinelli et al. 2019). Let \( Var \) 
be an infinite set of variables \( x_0, x_1, \ldots \) for referring to the 
strategies. A binding prefix over a set \( \Gamma \subseteq A \) of agents and 
a set \( V \subseteq Var \) of variables is a finite sequence \( \Gamma = \{ (x, i) \mid i \in \Gamma \} \) of 
length \( |\Gamma| = |\Gamma| \), such that every agent 
\( i \in \Gamma \) occurs exactly once in \( \Gamma \). However, the same variable 
\( x \in V \) can occur several times in \( \Gamma \), i.e., intuitively, the 
strategy denoted by \( x \) can be used by several agents in \( \Gamma \).

The binding operator \( (x, i) \) means that strategy \( x \) is used by 
agent \( i \). A quantification prefix over a set \( V \subseteq Var \) is a 
finitne sequence \( \varphi \in \{ \exists x, \forall x \mid x \in V \}^{V[P]} \) of length \( |\varphi| = |\varphi| \) 
such that each variable \( x \in V \) occurs in \( \varphi \) exactly once. The 
strategy quantifier \( \exists x \) (resp. \( \forall x \)) reads as “for some (resp. 
every) strategy \( x, \ldots \)”. Let \( Qnt(\varphi) \) \( \subseteq \{ \exists x, \forall x \mid x \in V \}^{V[P]} \) and 
\( Bnd(\Gamma) \subseteq \{ (x, i) \mid i \in \Gamma \} \) denote the 
sets of all quantification and binding prefixes over variables 
\( V \) and agents in \( \Gamma \). For a given \( \varphi \), let \( E(\varphi) \) (resp. 
\( A(\varphi) \)) be the set of all agents \( i \) such that \( (x, i) \) \( \exists x \) 
( resp. \( \forall x \) ) appear in \( \varphi \) (i.e. the set of agents assigned 
to strategies under existential (resp. universal) quantification).

For example, \( \varphi = \forall x \exists y \forall z \in \{ (x, y, z) \} \), 
\( b = \{ (x, 1), (x, 2), (y, 3), (z, 4) \} \), 
\( Bnd(\Gamma) = \{ (x, 1), (x, 2), (y, 3), (z, 4) \} \), 
\( E(\varphi) = \{ 3 \} \) and \( A(\varphi) = \{ 1, 2, 4 \} \).

For a given formula \( \varphi \), the set \( free(\varphi) \) of \( free(\varphi) \)
free agents/variables was defined in (Mogavero et al. 2014) as 
the subset of \( A \cup Var \) containing (i) all agents for which 
there is no binding after the occurrence of a temporal operator 
and (ii) all variables for which there is a binding but 
no quantifications. Formally, \( free : SL[SG] \to 2^{A \cup Var} \) 
is a function defined as follows: (i) \( free(p) = \emptyset \), where 
\( p \in PV \); (ii) \( free(\neg \varphi) = free(\varphi) \); (iii) \( free(\varphi_1 \land \varphi_2) = free(\varphi_1) \cup free(\varphi_2) \); (iv) \( free(\exists x \varphi) = A \cup free(\varphi) \); 
and \( free(\varphi_1 \land free(\varphi_2) = free(\varphi_1) \cup free(\varphi_2) = \emptyset \).

\( free(\varphi) \) where \( i \in A \) and \( x \in Var \). 

Definition 3 (SL[SG]). The formulæ of Strategy Logic 
with Simple Goals are defined inductively from the set of 
atomic propositions \( PV \), strategy variables \( Var \), agents \( A \), 
and \( Bnd(\Gamma) \). \( \psi \in Qnt(free(\varphi)) \):

\[
\varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid \exists x \varphi \mid \forall x \varphi \mid \varphi U \varphi \mid \varphi U \varphi U \varphi.
\]

According to the syntax rule, the SL[SG] formulæ \( \exists x \varphi \) 
and \( \forall x \varphi \) consist of a quantification prefix \( \varphi \), a bind-
ing prefix \( b \), the temporal operator next \( X \) or until \( U \), 
and the well-formed (sub)formula \( \varphi \). Standard abbreviations 
are used for other temporal operators like globally \( G \) and 
eventually \( F \). For a given set of all agents \( A \) of MAS, 
each binding prefix \( \varphi \) contains every agent of \( A \), and 
every agent is bound to exactly one variable which is quantified 
in \( \varphi \). Therefore, the length of every \( \varphi \) is \(|\varphi| \). The 
conditions on \( b \) and \( \varphi \) ensure that every SL[SG] formula \( \varphi \) is 
a sentence, i.e., \( free(\varphi) = \emptyset \). Unlike in ATL, the quanti-
ifiers over variables referring to strategies can be of different 
types and can appear in any order. Moreover, two or more 
strategies can be assigned to the same strategy, which is also 
not expressible in ATL, e.g., \( \exists x (i, x) \exists j (j, x) X p \). 
SL[SG] is less expressive than ATL, but still allows to express 
interesting properties. In particular, one can specify and 
verify distributed systems in which knowledge explicitly 
represented is the basis for decision making, strategy selection, 
and implementation. An example of the above is cooperation 
of autonomous vehicles with the environment in order to 
avoid collisions. Thus, for each possible environment \( e \) 
strategy there must be a vehicle \( (v) \) strategy such that no 
collision occurs for any behavior of other road users \( u \): 
\( \forall x e \exists x \forall x u (x, e) \land \exists x \forall u (x, u) G (\neg \text{collision}) \).

Another example is the verification of correctness and fairness 
of voting protocols, in particular ensuring that coercion in 
voting, which has a Stackelberg structure, can be avoided. 
That is to ensure that for every strategy of the coercer \( c \), 
there is a strategy of the voter \( (v) \) such that no matter what 
the environment \( (e) \) is doing, the voter will vote according 
to its belief and knowledge without being penalized: 
\( \forall x e \exists x \forall x u (x, e) \land \exists x \forall u (x, u) G (\neg \text{punished}) \).

Belardinelli et al. 2019).

\begin{itemize}
  \item **Strategies.** Formally, a memoryfull strategy for an agent 
  \( i \in A \), or \( i \)-strategy, is a function \( \sigma : St_i \to Act_j \).

  The set of all strategies, for all agents, in model \( M \), is denoted 
  as \( \Sigma(M) \). A joint strategy \( \sigma_A \) assigns a strategy to 
every agent \( i \in A \). Given a strategy \( \sigma \) for an agent \( i \), if a 
different agent \( j \) is such that the range of strategy \( \sigma \) is a subset of 
\( Act_j \) (i.e. \( ran(\sigma) \subseteq Act_j \)), then intuitively also agent \( j \) 
can use the strategy \( \sigma \). We say that such a strategy is shared 
by agents \( i \) and \( j \) and denote by \( shr(x, \sigma) \) the set of agents 
bounded to the variable \( x \) within the formula \( \sigma \). The set of 
all strategies shared by agents from \( shr(x, \sigma) \) is denoted
\end{itemize}
Given a model (an interpreted system) \( M \), being consistent with an incomplete (partial) model. In what follows, we introduce a model that assigns Boolean values to the variables and thus enables us to distinguish sub-formulae that are not inductively defined. The satisfaction relation \( h \) on all other variables and agents. Given a history \( h \in S \), an assignment \( \chi \) defines a unique computation \( \Lambda(h, \chi) = h \cdot (\xi(h))_{\in A} \cdot g_1 \cdot (\xi(h; g_1))_{\in A} \cdot g_2 \ldots \) starting with \( h \) and being consistent with \( \chi \).

**Semantics.** Given a model (an interpreted system) \( M \), we inductively define the satisfaction relation \( (M, h, \chi) \vdash \varphi \) where \( h \) is a history, \( \varphi \) is a formula, and \( \chi \) is an assignment. To give interpretation of formulae in a model, we need to distinguish sub-formulae that are not SL[SG] formulae by themselves and give semantics also for them. Let \( \mathcal{T} = \{ X, \varphi, \varphi \cup \varphi_2 : \varphi, \varphi_1, \varphi_2 \in \text{SL[SG]} \} \) - be a set of the temporal subformulae of the formulas of \( \text{SL[SG]} \).

\[
\begin{align*}
(M, h, \chi) & \vdash \varphi \quad \text{iff} \quad p \in \mathcal{P}(	ext{last}(h)), \text{for } p \in \mathcal{P}V
(M, h, \chi) & \vdash \neg \varphi \quad \text{iff} \quad (M, h, \chi) \not\vdash \varphi
(M, h, \chi) & \vdash \varphi \land \varphi_2 \quad \text{iff} \quad (M, h, \chi) \vdash \varphi_1 \text{ and } (M, h, \chi) \vdash \varphi_2
(M, h, \chi) & \vdash \exists \psi, \quad \text{iff} \quad \exists \sigma \in \mathcal{S}(M, \chi(s)), \\
& \quad \exists \psi, \quad \phi \in \mathcal{T}, \psi \in \mathcal{P} \vdash \sigma \vdash \psi
(M, h, \chi) & \vdash \forall \psi, \quad \text{iff} \quad \forall \sigma \in \mathcal{S}(M, \chi(s)), \\
& \quad \forall \psi, \quad \phi \in \mathcal{T}, \psi \in \mathcal{P} \vdash \sigma \vdash \psi
(M, h, \chi) & \vdash (x, i) \psi \quad \text{iff} \quad (M, h, \chi(x)) \vdash \psi, \\
& \quad \text{for } \psi \in \mathcal{T}, \phi \in \mathcal{T}, \psi \in \mathcal{P} \vdash \mathcal{A}(\chi(x))
(M, h, \chi) & \vdash X \varphi \quad \text{iff} \quad (M, \lambda(h, \chi), \chi(\lambda(1)) \vdash \varphi, \\
(M, h, \chi) & \vdash \varphi_1 \cup \varphi_2 \quad \text{iff} \quad \exists k \geq |h| \text{ s.t. } (M, \lambda(h, \chi)_{\leq k}, \chi) \vdash \varphi_2 \\
& \quad \text{and } (M, \lambda(h, \chi)_{< l}, \chi) \vdash \varphi_1, \forall l: |l| \leq k.
\end{align*}
\]

**Definition 4** (Model, satisfiability). We write \( (M, h, \chi) \vdash \varphi \) iff \( (M, h, \chi) \vdash \varphi \) for every assignment \( \chi \) and say that an interpreted system \( M \) is a model of an SL[SG] formula \( \varphi \), in symbols \( M \models \varphi \), if \( M, i \models \varphi \), i.e., \( \varphi \) is true at the initial state (the history of length 1) of the model \( M \). An SL[SG] formula \( \varphi \) is satisfiable iff there is a model for it.

We consider the problem of deciding whether a SL[SG] formula is satisfiable under some fixed initial restrictions on MAS, concerning the number of agents, local actions, local states, and local propositions of every agent. Therefore, we deal with the bounded satisfiability problem. Since SL[1G] has the Bounded Model Property (Mogavero et al. 2017), so SL[SG] (a sublogic of SL[1G]), inherits this property. We call this decision problem SL[SG] SAT.

### 4 Boolean Representation of SL[SG] Models

In this section we show how to encode a MAS and its models using a set of fresh propositional variables. For a given model, the number of variables is fixed. A function is determined that assigns Boolean values to the variables and thus encodes all elements of the model. Then, to find this model, it is enough to determine the valuation function. The procedure looking for this function assigns values to variables step by step. Thus, until the model is found, we are dealing with an incomplete (partial) model. In what follows, we show how we define and encode partial models, and extend them to total models which are an input for a model checker.

### 4.1 Encoding a MAS

Consider a MAS as defined in Section 1. We assume that for each agent \( i \in A \) the i-local states are numbered from 1 to \( |L_i| \), and the joint actions are numbered from 1 to \( |\mathcal{A}x| \). Below, we define sets of fresh propositional variables: \( \mathcal{P}B_i, \mathcal{P}B_i, \mathcal{V}B_i \) for \( i \in A \), and let \( \mathcal{V}B = \bigcup_{i \in A} \mathcal{V}B_i \) and \( \mathcal{P}B = \bigcup_{i \in A} \{TB_i \cup PB_i \cup VB_i\} \). The semantics of these variables is defined by the valuation function: \( v_M : \mathcal{P}B \rightarrow \{0, 1\} \), described after the definition of each set of the propositional variables of \( \mathcal{P}V \). Next, we allocate the above mentioned fresh variables:

- For the i-local protocols for \( i \in A \):

  \[ P_B = \{ pb_i(k, t) \mid 1 \leq k \leq |L_i|, 1 \leq t \leq |\mathcal{A}x| \} \]

- For the i-local transition functions for \( i \in A \):

  \[ TB_i = \{ tb_i(k, j, k') \mid 1 \leq k, k' \leq |L_i|, 1 \leq j \leq |\mathcal{P}V_i| \} \]

- For the i-local valuation functions for \( i \in A \):

  \[ VB_i = \{ vb_i(k, j) \mid 1 \leq k \leq |L_i|, 1 \leq j \leq |\mathcal{P}V_i| \} \]

Next, we encode the properties satisfied by the components of a MAS.

**A1** For each \( i \in A \) and \( l_i \in L_i \): \( P_l(i) \neq \emptyset \). This is encoded by the following formula \( \varphi_1 = \bigwedge_{n; k \leq n} \bigvee_{t \leq m} pb_i(k, t) \).

**A2** Let \( gl_i : \{1, \ldots, |\mathcal{A}x|\} \rightarrow 2^{\{1, \ldots, |\mathcal{A}x|\}} \), for \( i \in A \), be a function that assigns global action numbers to a set of numbers of the joint actions s.t. \( t \) is the number of their \( i \)-th components. That is: \( \{ gl_i(t) = 1 \leq j \leq |\mathcal{A}x| \mid \alpha_i = a_i^j \} \).

For each agent \( i \in A \), \( T_i(t, \alpha) \) is defined iff \( \alpha \in P_l(i) \). This is encoded by the following formula \( \varphi_2 = \bigwedge_{i \leq n; m, k \leq n} ( \bigvee_{t \leq m} t b_i(k, j, k') \leftrightarrow p b_i(k, t) ) \).

**A3** For each agent \( i \in A \), \( T_i \) is a function. This is encoded by the following formula \( \varphi_3 = \bigwedge_{i \leq n; m, k \leq n; j \leq |\mathcal{A}x|} ( t b_i(k, j, k') \rightarrow \bigvee_{k'' \leq n, k'' \neq k'} t b_i(k, j, k'') ) \).

### 4.2 Encoding Models

A valuation function \( v_M \) which satisfies the conditions C1-C3 and the formula \( \varphi_1 \land \varphi_2 \land \varphi_3 \), determine the model \( M \) of MAS without an initial state specified. Therefore, \( v_M \) actually encodes a family of the models that differ only in the initial state. The set of global states is a product of the local state sets. Then, given some \( v_M \), we can reconstruct the (partial) global transition function \( T \) using the following fact: \( T(g, a) = g' \) iff \( \forall t \leq m : v_M(t b_i(g, a, g')) = 1 \). Finally, the valuation function \( V \) can be easily reconstructed using the observation: \( p^j \in \mathcal{V}(g) \) iff \( v_M(vb_i(g, j)) = 1 \). Thus, searching for a model for any formula \( \psi \) can be then reduced to searching for a valuation function \( v_M \).

403
4.3 Partial MAS and Partial Models

Notice that the i-local components of each agent \(i \in A\) can be rewritten as follows. Let \(B = \{0, 1\}\):

- \(P_i: L_i \times Act_i \rightarrow B\) s.t. \(\forall l \in L, \exists \alpha \in Act, P_i(l, \alpha) = 1\);
- \(T_i: L_i \times Act \times L_i \rightarrow B\) s.t. \(\forall l \in L, \exists \alpha, \alpha' \in Act, (P_i(l, \alpha') = 1\) iff \(\exists l' \in L, T_i(l, \alpha, l') = 1\) and if \(T_i(l, \alpha, l') = 1\), then \(\forall l'' \in L, \forall \alpha'' \neq \alpha', T_i(l, \alpha'', l'') = 0\).
- \(V_i: L_i \times \mathcal{P}V_i \rightarrow B\).

The above conditions are the same as \(A1-A3\). The partial versions \(PP_i, PT_i, PV_i\) of the i-local components of each agent \(i \in A\) are defined by extending the set of values of the functions \(P_i, T_i, V_i\) to the set \(B_u = \{0, 1, u\}\), where \(u\) stands for undefined. Formally:

- a partial protocol function \(PP_i: L_i \times Act_i \rightarrow B_u\),
- a partial transition function \(PT_i: L_i \times Act \times L_i \rightarrow B_u\),
- a partial valuation function \(PV_i: L_i \times \mathcal{P}V_i \rightarrow B_u\).

By a partial MAS, denoted \(\mathcal{M}_P\), we mean a MAS in which each agent \(i\) is associated with the i-local partial components.

A partial MAS\(_P\) induces the partial model \(M_P\), which differs from a model \(M\) for a MAS in the global transition function and the global valuation function, called partial, and defined as follows:

- \(PT: St \times Act \times St \rightarrow B_u\) such that \(PT(g, i, g') = 1\) iff \(PT_i(g, i, g') = 1\) for all \(i \in A\), \(PT(g, i, g') = u\) iff \(PT_i(g, i, g') = u\) for some \(i \in A\), \(PT(g, i, g') = 0\) otherwise,
- \(PV: St \times \mathcal{P}V \rightarrow B_u\) such that \(PV((l_1, \ldots, l_n), p) = PV_i(l_i, p)\), where \(p \in \mathcal{P}V_i\) for \(i \in A\).

Each partial model \(M_P\) can be represented by the partial valuation function \(v_{M_P}: \mathcal{P}V\rightarrow B_u\). The goal of our satisfiability algorithm is to find values of \(B\) for the undefined variables and to extend \(v_{M_P}\) to a total function satisfying the conditions \(C1-C3\) and the formulae \(A1-A3\).

4.4 Construction of \(M_{\text{over}}^\Gamma\) and \(M_{\text{under}}^\Gamma\)

We construct under and over approximations of partial models. For each partial model \(M\) satisfying \(A1-A3\) and \(\Gamma \subseteq A\), (total) models \(M_{\text{under}}^\Gamma\) and \(M_{\text{over}}^\Gamma\) are constructed. First, for every agent \(i \in A\) we define a necessary local protocol: \(P_i: L_i \times Act_i \rightarrow B\), a necessary local transition function \(T_i: L_i \times Act \times L_i \rightarrow B\), and a necessary local valuation function \(V_i: L_i \times \mathcal{P}V_i \rightarrow B\), a possible local protocol \(\overline{P_i}: L_i \times Act_i \rightarrow B\), a possible local transition function \(\overline{T_i}: L_i \times Act \times L_i \rightarrow B\), a possible local valuation function \(\overline{V_i}: L_i \times \mathcal{P}V_i \rightarrow B\), where:

- if \(PP_i(l, a) \neq u\), then \(\overline{P_i}(l, a) = \overline{P_i}(l, a) = PP_i(l, a),\)
- if \(PT_i(l, \alpha, l') \neq u\), then \(\overline{T_i}(l, \alpha, l') = \overline{T_i}(l, \alpha, l') = PT_i(l, \alpha, l'),\)
- if \(PV_i(l, p) \neq u\), then \(\overline{V_i}(l, p) = \overline{V_i}(l, p) = PV_i(l, p),\)
- if \(PP_i(l, a) = u\), then \(\overline{P_i}(l, a) = 0, \overline{P_i}(l, a) = 1,\)
- if \(PT_i(l, \alpha, l') = u\), then \(\overline{T_i}(l, \alpha, l') = 0, \overline{T_i}(l, \alpha, l') = 1,\)
- if \(PV_i(l, p) = u\), then \(\overline{V_i}(l, p) = 0, \overline{V_i}(l, p) = 1.\)

Notice that the possible local protocol is an extension of the necessary local protocol, i.e., \(\forall l \in L, \alpha \in Act, \overline{P_i}(l, a) \leq \overline{P_i}(l, a)\) and the possible local transition function is an extension of the necessary local transition function, i.e., \(\forall l \in L, \alpha \in Act, \overline{T_i}(l, \alpha, l') \leq \overline{T_i}(l, \alpha, l')\).

Similarly, total valuations of the propositional variables are defined: a necessary valuation \(\overline{V_i}: St \times \mathcal{P}V \rightarrow B\) and a possible valuation \(\overline{V_i}: St \times \mathcal{P}V \rightarrow B\) such that:

- if \(PV(g, p) \neq u\), then \(\overline{V_i}(g, p) = \overline{V_i}(g, p) = PV(g, p),\)
- if \(PV(g, p) = u\), then \(\overline{V_i}(g, p) = 0, \overline{V_i}(g, p) = 1.\)

Thus, for every \(g \in St, p \in \mathcal{P}V\) we have \(\overline{V_i}(g, p) \leq V_i(g, p)\).

The (total) model \(M_{\text{under}}^\Gamma\) is defined as in Def. 2 for the MAS, which consists of \(n\) agents, where each agent \(i \in \Gamma\) is associated with \(AG_i = (L_i, i, Act_i, P_i, \overline{P_i}, PV_i, V_i)\), and each agent \(j \in A \setminus \Gamma\) is associated with \(AG_j = (L_j, i, Act_j, \overline{P_i}, \overline{P_i}, PV_j, V_j)\). The (total) model \(M_{\text{over}}^\Gamma\) is defined as in Def. 2 for the MAS, which consists of \(n\) agents, where each agent \(i \in \Gamma\) is associated with \(AG_i = (L_i, i, Act_i, PP_i, PT_i, \overline{P_i}, PV_i, V_i)\), and each agent \(j \in A \setminus \Gamma\) is associated with \(AG_j = (L_j, i, Act_j, PP_i, \overline{P_i}, PV_j, V_j)\).

**Definition 5.** Let \(M = (St, t, PT, PV)\) and \(M' = (St, t, PT', PV')\) be two (partial) models for the partial MAS and MAS', consisting of \(n\) agents, where each agent \(i \in A\) is associated with \(AG_i = (L_i, i, Act_i, PP_i, PT_i, \overline{P_i}, PV_i, V_i)\) and \(AG'_i = (L_i, i, Act_i, PP_i', PT_i', \overline{P_i}, PV_i, V_i)\), resp. We say that \(M\) extends \(M'\) iff the following conditions hold:

- if \(PP'_i(l, a) \neq u\), then \(PP_i(l, a) = PP'_i(l, a),\)
- if \(PT'_i(l, \alpha, l') \neq u\), then \(PT'_i(l, \alpha, l') = PT'_i(l, \alpha, l'),\)
- if \(PV'(g, p) \neq u\), then \(PV(g, p) = PV'(g, p).\)

Notice that for a given partial model \(M\) and \(\Gamma \subseteq A\), both total models \(M_{\text{under}}^\Gamma\) and \(M_{\text{over}}^\Gamma\) extend \(M\).

**Theorem 1.** Consider two partial models \(M\) and \(M'\) s.t. \(M\) extends \(M'\), and let \(\Gamma \subseteq A\). The following conditions hold:

- \(v_{M_{\text{under}}^\Gamma}(b_i) \leq v_{M_{\text{under}}^\Gamma}(b_i) \leq v_{M_{\text{over}}^\Gamma}(b_i)\) for \(b_i \in TB \cup PB\) with \(i \in \Gamma\),
- \(v_{M_{\text{under}}^\Gamma}(b_i) \geq v_{M_{\text{under}}^\Gamma}(b_i) \geq v_{M_{\text{over}}^\Gamma}(b_i)\) for \(b_i \in TB \cup PB\) with \(i \in A \setminus \Gamma\),
- \(v_{M_{\text{under}}^\Gamma}(v_i) \leq v_{M_{\text{under}}^\Gamma}(v_i) \leq v_{M_{\text{over}}^\Gamma}(v_i) \leq v_{M_{\text{over}}^\Gamma}(v_i)\) for \(v_i \in V\).

Proof. From the definitions of over and under models.

5 Monotonic Theory for SL[SG]

Our satisfiability algorithm for SL[SG] applies the technique used earlier for monotonic theories for checking both CTL satisfiability (Bayless et al. 2015; Klenke, Bayless, and Hu 2016) and ATL satisfiability (Kapcprzak, Niewiadomski, and Penczek 2020). Although SL[SG] is not a monotone theory, we show how the same technique can be applied in this case. In what follows, specific cases of positive or negative monotonicity are described. These properties are used to determine the class of models to which the model of a given formula belongs. This makes it possible to substantially limit the set of all potential models, only to models
belonging to this class and significantly reduce the search space for optimizing the satisfiability procedure.

5.1 Boolean Monotonic Theory

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

The definition of (positive and negative Boolean) monotonicity for a function $F : B^n \rightarrow 2^S$ (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 $1 \leq i \leq n$. 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 $1 \leq i \leq n$.

Definition 6 (Boolean Monotonic Theory). A theory $T$ with a signature $\Omega = (S, S_f, S_r, A_r)$, 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 $A_r$ is the arity of the relation and function symbols, is (Boolean) monotonic iff all sorts in $\Omega$ are Boolean, and all predicates and functions in $\Omega$ are monotonic.

5.2 Monotonicity Property

The satisfaction relation $|=\text{for SL}[SG]$ is not monotonic, but satisfies the condition of monotonicity in some special cases.

Theorem 2. Let $M$ and $M'$ be two models defined over the same set of agents $A$, for two MAEs $\mathcal{A}$ that share the same sets $L_i, \text{Act}_i, \mathcal{P}_i \forall i \in A$. Then, we have $(M, h, \chi) |= \phi$ implies $(M', h, \chi) |= \phi$, for any $h$ and $\chi$, if the following conditions hold:

- (H1) $\phi \in \{p, p_1 \wedge p_2\}$ for $p, p_1, p_2 \in \mathcal{P}_i$ and $v_M(v_b) \leq v_M(v_b) \text{ for each } v_b \in V_i$.
- (H2) $\phi = \neg p$ for $p \in \mathcal{P}_i$ and $v_M(v_b) \geq v_M(v_b) \text{ for each } v_b \in V_i$.
- (H3.1) $v_M(v_b) \leq v_M(v_b) \text{ for each } v_b \in V_i$, and $v_M(b_i) \leq v_M(b_i) \text{ for each } b_i \in TB_i \cup PB_i$.
- (H3.2) $v_M(b_i) \geq v_M(b_i) \text{ for each } b_i \in TB_i \cup PB_i$.
- (H3.3) $v_M(b_i) \in A(b_i)$.

Proof. Let $M$ and $M'$ be two models defined over the same set of agents $A$, for two multi-agent systems that share the same sets $L_i, \text{Act}_i, \mathcal{P}_i \forall i \in A$.

1. Let $\phi = p_1 \wedge p_2$. $h$ is a history, $\chi$ is an assignment, and $p_1 \in \mathcal{P}_i, p_2 \in \mathcal{P}_j$ for some $i, j \in A$. Assume that $(M, h, \chi) |= p_1 \wedge p_2$, i.e. $p_1, p_2 \in V_i(\text{last}(h))$. Thus $v_M(v_b) = v_M(v_b) = v_M(v_b)$.

2. Let $\phi = \neg p$. $h$ is a history, and $p \in \mathcal{P}_i$ for some $i \in A$. Assume that $(M, h, \chi) = \neg p$, i.e. $p \notin V_i(\text{last}(h))$. Thus $v_M(v_b) = v_M(v_b)$.

3. Let $\phi = \forall \chi X p$. $h$ is a history, and $p \in \mathcal{P}_i$ for some $i \in A$. Denote by $E$ ($A$) the set of all variables that appear under existential (universal, resp.) quantifier in $\phi$. Assume that $(M, h, \chi) = \phi$. Thus for every $h \in E$ there exists a strategy $s_i$ and for every $p \in A$, for every strategy $s_i \in A$, $(M, h, \chi') = \chi \text{ for } s_i \text{ being an assignment that maps } x \rightarrow s_i \text{, i.e. } s_i \in \text{shr}(x, \phi)$, and coincides with $\chi$ on all other variables. Thus $(M, h, \chi) = \phi$, i.e. $p \notin V_i(\text{last}(\lambda h, \chi') \leq |i| + 1)$.

On the other hand, assume that $(M', h, \chi') = \phi$. Thus (*) in both models $M$ and $M'$ there exist the same strategies and $\text{last}(\lambda h, \chi') \leq |i| + 1 = \text{last}(\lambda h, \chi') \leq |i| + 1$, but $p \notin V_i(\phi)$ or $p \notin V_i(\phi)$, or (** in both models $M$ and $M'$ there exist the same strategies, but $\text{last}(\lambda h, \chi') \leq |i| + 1 = \text{last}(\lambda h, \chi') \leq |i| + 1$, or (*** the set of strategies that can be assigned to agents $i \in E(\phi) \in M$ (denoted $\Sigma(\Sigma(E(\phi)))$) is a subset of the set of strategies that can be assigned to these agents in $M'$ (denoted $\Sigma(\Sigma(E(\phi)))$), and the set of strategies that can be assigned to agents $j \in A(\phi)$ in $M$ (denoted $\Sigma(A(\phi)\Sigma(E(\phi)))$) is a subset of the set of strategies that can be assigned to these agents in $M'$ (denoted $\Sigma(A(\phi)\Sigma(E(\phi)))$) the condition $(M', h, \chi') = \phi$ holds, or $\Sigma(A(\phi)) \subseteq \Sigma(\Sigma(E(\phi)))$, i.e., the set of strategies that can be assigned to agents $i \in E(\phi)$ in $M'$ is a proper subset of the set of strategies that can be assigned to these agents in $M$. Let us consider all the above cases.

If (*) then $v_M(v_b) = v_M(v_b)$, what contr. (H3.1). If (** then

(a) for some $i \in E(\phi)$, let $l_i = \text{last}(\lambda h, \chi') \leq |i| + 1$ and $l'_i = \text{last}(\lambda h, \chi') \leq |i| + 1$.

Then, $v_M(v_b) = v_M(v_b)$, and $v_M(v_b) = v_M(v_b)$ for a global action $\alpha$ s.t. $\alpha^k$ for $k \in A$ is determined by the strategy assigned to $b$ by $\chi'$. Thus from A1-A3 (action determination) $v_M(v_b) = v_M(v_b)$, what contradicts (H3.2).

(b) for some $j \in A(\phi)$, let $l_j = \text{last}(\lambda h, \chi') \leq |i| + 1$ and $l'_j = \text{last}(\lambda h, \chi') \leq |i| + 1$.

Then, $v_M(v_b) = v_M(v_b)$, and $v_M(v_b) = v_M(v_b)$ for a global action $\alpha$.
s.t. $\alpha^k$ for $k \in A$ is determined by the strategy assigned to $k$ by $\chi'$. Thus from A1-A3 (action determinism) $v_M(t_b((last(h), \alpha, t'_j))) = 0$. Therefore, $v_M(t_b((last(h), \alpha, t'_j))) < v_M(t_b((last(h), j, \alpha, t'_j)))$ what contradicts (H3.3).

If (***) then for some $j \in A(\phi)$ there exists a strategy $\sigma_j$ in $M'$ which does not exist in $M$. It means that there exists an action $\alpha^k$ which is possible to execute in $last(h)$, in $M'$ but not in $M$. Thus, $v_M(p_b((last(h), j, k))) = 0$ and $v_M(p_b((last(h), j, k))) = 1$. Therefore, $v_M(p_b((last(h), j, k))) = v_M(p_b((last(h), j, k)))$ what contradicts (H3.3).

If (****) then for some $i \in E(\phi)$ there exists a strategy $\sigma_i$ in $M$ which does not exist in $M'$. It means that there exists an action $\alpha^k$ which is possible to execute in $last(h)$, in $M$ but not in $M'$. Thus, $v_M(p_b((last(h), k))) = 1$ and $v_M(p_b((last(h), k))) = 0$. Therefore, $v_M(p_b((last(h), k))) > v_M(p_b((last(h), k)))$ what contradicts (H3.2).

Finally, $(M, h, \chi) \models \phi$ implies $(M', h, \chi) \models \phi$. The proof for $\phi(p_1 U p_2)$ is analogous.

It follows from Thm 2 that the relation $\models$ is positive monotonic w.r.t. $VB$ for $\phi \in \{p, p_1 \land p_2, \phi X p, \phi(p_1 U p_2)\}$, negative monotonic w.r.t. $VB$ for $\phi \in \{\neg p\}$, positive monotonic w.r.t. $TB_1$ and $PB_1$ for $\phi \in \{\phi X p, \phi(p_1 U p_2)\}$ and $i \in E(\phi)$, negative monotonic w.r.t. $TB_1$ and $PB_1$ for $\phi \in \{\phi X p, \phi(p_1 U p_2)\}$ and $i \in E(\phi)$.

5.3 Function MC: Model Checking

Let $\|\phi\|_M$ denote the set of all states of $M$ (i.e., histories of length 1), where $\phi$ holds. In order to compute $\|\phi\|_M$, the evaluation function $MC(op, Z_1, V_m)$ is defined for an unary operator $op$, and $MC(op, Z_1, Z_2, V_m)$ for a binary operator $op$, where $Z_1, Z_2 \subseteq St$, and $V_m$ is a valuation encoding a model. The function $MC$ evaluates the operator $op$ on the sets of states $Z_1, Z_2$ rather than the formulae holding in these states. If $\phi = p \in PV$, then $\|\phi\|_M$ returns the set of states of $M$ in which $p$ holds, i.e., $\{s \in St \mid p \in V(g)\}$. Otherwise, $\|\phi\|_M$ takes the top-most operator $op$ of $\phi$ and solves its argument(s) recursively using the function $MC$ and applying $MC(op, Z_1, V_m)$ or $MC(op, Z_1, Z_2, V_m)$ to the returned set(s) of states. Notice that $MC(\neg, Z, V_m)$ returns $\neg St \setminus Z$. Similarly, $MC(\land, Z_1, Z_2, V_m)$ returns $Z_1 \cap Z_2$. If $Op \in \{\phi X, \phi U\}$, then the model checking algorithm of (Belardinelli et al. 2019) is used. The algorithm manipulates sets of states in $St$ that is inspired by the standard model checking algorithm for ATL. It works bottom-up on the structure of a formula. Its soundness and completeness is guaranteed by the fixed-point characterisation of SL[SG]. Since this algorithm works on sets of states, our approximation procedure also works on sets of states. One can show that for SL[SG] with complete information both semantics (defined on states and histories) are equivalent.

Now, Thm 2 can be rewritten by replacing the propositional variables by the sets of states these variables hold at.

**Theorem 3.** The functions $MC(op, Z_1, V_m)$ and $MC(op, Z_1, Z_2, V_m)$ are:

1. negative monotonic w.r.t. $VB$ and positive and negative monotonic w.r.t. $PB_1$ and $TB_1$ for $i \in A$, for $op = \land$,
2. positive monotonic w.r.t. $VB$ and positive and negative monotonic w.r.t. $PB_1$ and $TB_1$ for $i \in A$, for $op = \lor$,
3. positive monotonic w.r.t. $VB$, positive monotonic w.r.t. $PB_1$ and $TB_1$ for $i \in E(\phi)$, negative monotonic w.r.t. $PB_i$ and $TB_i$ for $i \in A(\phi)$, for $op \in \{\phi X, \phi U\}$.

6. Approximations

Next we show a technique for constructing an approximation for a given incomplete model derived from the initial requirements. This technique uses partial model extension methods and monotonicity properties, described earlier. It allows to determine the class of possible extensions of a partial model including all models satisfying the given formula. For a given partial model $M'$ and a formula $\phi$ two approximations are computed:

- approximation of models - it is a set of models, denoted $CLASS_M'(\phi)$, such that if $M$ is a total extension of $M'$ and $s \models \phi$ for some $s \in St$, then $M \in CLASS_M'(\phi)$;
- states - it is a set of states, denoted by $\|\phi\|_M'$, containing all the states satisfying $\phi$ for some model of $CLASS_M'(\phi)$ i.e., $\{s \in St \mid \exists M \in CLASS_M'(\phi)$ s.t. $M, s \models \phi\} \subseteq \|\phi\|_M'$.

Notice that the above notion $\|\phi\|_M'$ for partial models $M'$ extends the same notion defined before for models.

Since $CLASS_M'(\phi)$ and $\|\phi\|_M'$ are only approximations, models and states that do not satisfy the formula can also be included. Furthermore, for a fixed partial model $M'$, a formula $\phi$, and an initial state $\iota$, it follows directly from the definitions that if $M'$ can be extended to a total model $M$ s.t. $M, \iota, \chi \models \phi$, then $M \in CLASS_M'(\phi)$ and $\iota \in \|\phi\|_M'$. The approximations are used in the satisfiability procedure we provide, for the early detection of conflicts and thus for finding the model faster. If the fixed initial state does not belong to the determined state approximation, then the model class determined by this approximation can be rejected, which greatly improves the efficiency of the procedure.

6.1 Algorithm Apx

The approximation of states is computed by the algorithm $Apx$ - see Algorithm 1. Its input contains: an $SL[SG]$ formula $\phi$, the partial model $M$ for a partial MAS, and a parameter $\lambda$, where $\lambda \in \{over, under\}$. The algorithm returns the $\lambda$-approximation of a set of states satisfying $\phi$. In what follows, we use the following notation: $\bar{\lambda} = over$ for $\lambda = under$, and $\overline{\lambda} = under$ for $\lambda = over$.

If $\phi = p$ is a propositional variable, then $Apx(\phi, M, \lambda)$ returns a set of states satisfying $p$ in the model $M^\lambda$. If $\phi = \neg \varphi$, then $Apx(\phi, M, \lambda)$ computes the complement of the approximation $Apx(\varphi, M, \overline{\lambda})$ of $\varphi$. If $\phi = \psi X \varphi$, then $Apx(\phi, M, \lambda)$ computes the approximation for $\varphi$ performing $Z = Apx(\varphi, M, \lambda)$ and the function $MC$ determines the set of states satisfying $\psi X \varphi$ in $M^\lambda$, where $\kappa Z$ is a fresh propositional variable that holds in the states of $Z$. In fact, $MC$ has a set of states as its input, not a formula. This
Algorithm 1 Algorithm $Apx$

**Input:** $\phi, \lambda, M$ for a partial MAS

**Output:** a set of states of $M$

1. if $\phi \in \mathcal{PV}$ then
2. $\ return \ {g \in S_t \mid M^A, g \models \phi}\$
3. else if $\phi = op(\psi)$ then
4. if $\ op = \neg$ then // negative monotonic
5. $Z := Apx(\psi, M, \lambda)$
6. return $S_t \setminus Z$
7. else if $op = \phi X$ then
8. $Z := Apx(\psi, M, \lambda)$
9. return $MC(op, Z, v_{M^A})$
10. else if $\phi = op(\psi_1, \psi_2)$ for $op \in \{\phi \cup, \land\}$ then
11. $Z_1 := Apx(\psi_1, M, \lambda)$
12. $Z_2 := Apx(\psi_2, M, \lambda)$
13. if $op = \phi \cup$ then
14. $\ return \ MC(op, Z_1, Z_2, v_{M^A})$
15. else if $op = \land$ then
16. $\ return \ Z_1 \cap Z_2$

Since $Z \subseteq Z'$ we have $MC(op, Z, v_{M^A}) \subseteq MC(op, Z', v_{M^A})$. Next, $MC(op, Z', v_{M^A}) \subseteq MC(op, Z', v_{M^A})$ from Thm 1 and Thm 3. Finally $MC(op, Z, v_{M^A}) \subseteq MC(op, Z', v_{M^A})$ and $Apx(\phi, M, over) \subseteq Apx(\phi, M', over)$ from def. of $Apx$. If $Z = Apx(\psi, M, under)$ and $Z' = Apx(\psi, M', under)$, then $Z' \subseteq Z$ from IA. Observe that $Apx(\phi, M, over) = MC(op, Z, v_{M^A})$. Similarly, $Apx(\phi, M', over) = MC(op, Z', v_{M^A})$. Since $Z' \subseteq Z$ we have $MC(op, Z', v_{M^A}) \subseteq MC(op, Z, v_{M^A})$ from Thm 1 and Thm 3. Finally $MC(op, Z', v_{M^A}) \subseteq MC(op, Z, v_{M^A})$ and $Apx(\phi, M', under) \subseteq Apx(\phi, M, under)$ from def. of $Apx$. \qed

6.2 Approximation of States

For a total model $M$ extending a partial model $M'$, the algorithm $Apx$ computes over and under-approximation of $\|\phi\|_M$. More precisely, $Apx(\phi, M', over)$ returns a set of states, which is an over-approximation of $\|\phi\|_M$. This means that if $t \in \|\phi\|_M$, then $t \in Apx(\phi, M', over)$. So, if $t \notin Apx(\phi, M', over)$, then there is no model $M$ extending $M'$ s.t. $t \models \phi$. Similarly, $Apx(\phi, M, under)$ computes an under-approximation of $\|\phi\|_M$, i.e., if $t \notin Apx(\phi, M', under)$, then $t \notin \|\phi\|_M$.

Theorem 5. Let $M'$ be a partial model and $M$ be a total model extending $M'$. Then, for a formula $\phi$ we have: $Apx(\phi, M, under) \subseteq \|\phi\|_M \subseteq Apx(\phi, M', over)$. \qed

7 Satisfiability Procedure

The $Apx$ algorithm and its property of Theorem 5 are used below to provide a new method for testing the SAT of SL[S[G] formulae and constructing their models.

7.1 Satisfiability Algorithm

The goal of the algorithm is to build a model for a given formula. Initially, the model requirements are established: the number of agents, the number of local actions of every agent, the number of local states of every agent, the number of local propositions of every agent, other requirements such as conditions A1-A3.

These requirements are given at the entry to the algorithm along with the SL[S[G] formula $\phi$ and the initial state $t$. \n
**Input:** $\phi, t$, and model requirements determining partial model $M$.

**Output:** total model $M$ s.t. $M, t \models \phi$, meeting the model requirements or the answer that such a model does not exist.

**Steps of the algorithm:** Let $depth$ be an integer variable tracking the decision depth of the solver.

1. $depth := 0$; set $v_M$ and check whether $v_M$ is consistent with the conditions A1-A3, if yes, then go to step 2, if no, then return UNSAT
2. compute $Apx(\phi, M, over)$
3. if \( \nu \in \text{Apx}(\phi, M, \text{over}) \), then
3.1 if \( M \) is total; SAT, return \( M \)
3.2 otherwise \( \text{depth} := \text{depth} + 1 \); assign value to an unsigned variable (the assignment must be consistent with the conditions A1-A3); a new partial model \( M \) is determined, restricting the class of the considered models; go to step 2
4. if \( \nu \not\in \text{Apx}(\phi, M, \text{over}) \), then
4.1 if \( \text{depth} > 0 \), analyse the conflict, undo decisions up to the conflict depth \( c \) and assign the opposite value to the conflicting variable; \( \text{depth} := c \); a new partial model \( M \) is determined; go to step 2
4.2 if \( \text{depth} = 0 \) return UNSAT.

The main idea behind the algorithm is to build a model step by step. The models are encoded with Boolean variables, so finding a model consists in determining their values. Some of them are determined by the initial model requirements and allow for establishing a certain class of models (model approximation). Using the monotonicity property (see Thm 4), the algorithm verifies whether the model in question may belong to this class. If so, the procedure narrows down the class of models. If not, it designates a new model class. The algorithm stops when either the model satisfying \( \phi \) is built or there is no model class to which it may belong. If a model satisfying \( \phi \) exists, then after the algorithm stops, the model approximation contains only a model satisfying \( \phi \) and the state approximation contains all states of this model that satisfy \( \phi \) (see Thm 5).

**Example 2.** Consider a MAS consisting of 2 agents, 2 local states and actions, and one local proposition per each agent. When the algorithm starts, none of the variables encoding local protocols, transitions, and propositions are assigned. Thus, all transitions and propositions values are possible at this step, as indicated by grey elements in Fig. 2 (left). Fig. 2 (right) presents the result of our algorithm - a MAS inducing a model for the formula \( \varphi_1 \) from Sec. 7.3. In this case all \( PB_1 \) and \( PB_2 \) variables have been assigned true and, thus, for every agent and local state, there is a transition labelled by every possible local action.

### 7.2 SGSAT Tool

We have implemented our algorithm on the top of MonoSat (Bayless et al. 2015) platform obtaining a prototype tool SGSAT confirming our theoretical results. The core of the system is a modified MiniSat solver (Eén and Sörensson 2003) extended to enable an implementation of monotonic theories, which results in an application similar to lazy-SMT solvers (Sebastiani 2007). In this case, we implemented the SL[SG] theory solver which makes use of an external STV model-checker (Kurpiewski, Jamroga, and Knapik 2019). Our tool reads model requirements and allocates SAT-solver variables (encoding a set of MASes) that are tracked by the theory solver. The SAT core, according to the CDCL algorithm, proposes assignments to the successive variables. The obtained values are propagated by the theory solver, which (using the STV tool) checks if the proposed valuation is consistent. If so, we continue assigning values to the variables until we have a complete model. If not, we get a conflict (a partial model that cannot be extended to a satisfactory valuation) which is analysed, the incorrect decisions are backtracked, and the solver proposes another assignment.

### 7.3 Experimental Results

Table 1 shows the preliminary experimental results. The column meaning, from left, the number of: agents, local states, local actions, local propositions, and variables encoding MAS. Next, the time consumed by SAT-solver, and the total runtime (in seconds).

<table>
<thead>
<tr>
<th>n</th>
<th>ls</th>
<th>la</th>
<th>lp</th>
<th>vars</th>
<th>( \varphi_1 ) satT</th>
<th>runT</th>
<th>( \varphi_2 ) satT</th>
<th>runT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>48</td>
<td>0.05</td>
<td>1.69</td>
<td>0.05</td>
<td>1.14</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>228</td>
<td>0.43</td>
<td>6.08</td>
<td>0.4</td>
<td>6.26</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>354</td>
<td>3.88</td>
<td>29.9</td>
<td>3.83</td>
<td>29.6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>1542</td>
<td>66.9</td>
<td>607</td>
<td>79.8</td>
<td>708</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>288</td>
<td>13.6</td>
<td>96.4</td>
<td>13.9</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1336</td>
<td>257</td>
<td>1700</td>
<td>270</td>
<td>2462</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>680</td>
<td>501</td>
<td>4121</td>
<td>423</td>
<td>397</td>
</tr>
</tbody>
</table>

Table 1: Preliminary experimental results. The column meaning, from left, the number of: agents, local states, local actions, local propositions, and variables encoding MAS. Next, the time consumed by SAT-solver, and the total runtime (in seconds).

8 Conclusions

A new method and the tool SGSAT for checking satisfiability for SL[SG] formulae have been defined and implemented. Since this is the only tool for testing satisfiability of SL[SG] and SL is general, we could not compare the experimental results with others. Our future plan is to extend our approach to larger fragments of SL, in particular for SL with One Goal and SL with epistemic operators and incomplete information. As the technique we introduced involves an external model-checker, its efficiency and applicability depends on the capabilities of the model-checker applied.
Acknowledgments

The work of M. Kacprzak was supported by the Białystk University of Technology, as part of the research grant W/II-I/IT/1/2019 of the Faculty of Computer Science and funded by Ministry of Science and Higher Education, Poland. W. Penczek acknowledges support from Luxembourgish/Polish FNR/NBIR project STV and CNRS/PAS project PARTIES.

References


