Strategy logic with simple goals: Tractable reasoning about strategies
Francesco Belardinelli, Wojciech Jamroga, Vadim Malvone, Aniello Murano

To cite this version:

HAL Id: hal-02377403
https://hal.science/hal-02377403
Submitted on 23 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Strategy Logic with Simple Goals: Tractable Reasoning about Strategies

Francesco Belardinelli\textsuperscript{1,2}, Wojciech Jamroga\textsuperscript{3,4}, Damian Kurpiewski\textsuperscript{3}, Vadim Malvone\textsuperscript{2} and Aniello Murano\textsuperscript{5}

\textsuperscript{1} Imperial College London, UK
\textsuperscript{2} Université d’Evry, France
\textsuperscript{3} Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
\textsuperscript{4} Interdisciplinary Centre for Security, Reliability, and Trust, SnT, University of Luxembourg
\textsuperscript{5} Università degli studi di Napoli “Federico II”, Italy

francesco.belardinelli@imperial.ac.uk, w.jamroga@ipipan.waw.pl, d.kurpiewski@ipipan.waw.pl, vadim.malvone@univ-evry.fr, murano@na.infn.it

Abstract

In this paper we introduce Strategy Logic with Simple Goals (SL[SG]), a fragment of Strategy Logic that strictly extends Alternating-time Temporal Logic ATL by introducing arbitrary quantification over the agents’ strategies. Our motivation comes from game-theoretic applications, such as expressing Stackelberg equilibria in games, coercion in voting protocols, as well as module checking for simple goals. We prove that model checking SL[SG] is \(P\)-complete, the same as ATL. Thus, the extra expressive power comes at no computational cost as far as verification is concerned.

1 Introduction

Formal verification of multi-agent systems (MAS) has been a thriving area of investigation in the last two decades, which has led to a wealth of logics to specify the temporal, epistemic, and strategic capabilities of agents, including Alternating-time Temporal Logic [Alur et al., 2002], possibly enriched with strategy contexts [Laroussinie and Markey, 2015], irrevocable strategies [Ågotnes et al., 2007], or operators for individual and group knowledge [Jamroga and van der Hoek, 2004; Hoek and Wooldridge, 2003]; Coalition Logic [Pauly, 2002]; and Strategy Logic [Chatterjee et al., 2010; Mogavero et al., 2014]. Besides theoretical results, some model checking tools have also been developed [Alur et al., 2001; Cimatti et al., 2002; Kacprzak et al., 2008; Huang and van der Meyden, 2014; Lomuscio et al., 2015; Cermák et al., 2015; Cermák et al., 2018].

The verification of MAS generates a tension between two conflicting demands. On the one hand, we need an expressive language to capture subtle temporal, epistemic, and game-theoretic notions such as reachability, common and distributed knowledge, and solution concepts in games. On the other hand, we need formalisms with a tractable model checking problem, for which efficient verification algorithms can be implemented. The same tension appears in the seminal paper [Alur et al., 2002] that introduced Alternating-time Temporal Logic. Two variants of the logic are presented: ATL\textsuperscript{+} is rather expressive, but with 2\textit{EXPTIME} verification complexity. Its fragment ATL has less expressivity, but allows to model check strategic properties in polynomial time. This leads to a nice balance of expressivity and computational complexity – indeed, ATL model checking is supported by a number of tools [Alur et al., 2001; Kacprzak et al., 2008; Lomuscio et al., 2015]. The question is: can we use a language that is more expressive than ATL, and still retains its polynomial-time complexity of model checking? We answer the question affirmatively in this paper.

Our starting point is Strategy Logic (SL) [Mogavero et al., 2014], an expressive extension of ATL\textsuperscript{*} that allows to characterize sophisticated game-theoretic solution concepts, e.g., Nash equilibria, dominant strategies, subgame-perfect equilibria, etc. Unfortunately, its model checking complexity is non-elementary, and hence highly unlikely to enable efficient implementation. Even restricted variants of SL, such as SL with memoryless strategies [Cermák et al., 2014], nested-goal, Boolean-goal, and one-goal SL [Mogavero et al., 2012] do not help much, as their model checking problems range from 2\textit{EXPTIME}-complete to non-elementary [Mogavero et al., 2014; Bouyer et al., 2015; Gardy et al., 2018].

We then propose to take one-goal SL, and further restrict goals to simple LTL formulas of type \(X \phi\) (next \(\phi\)), \(\phi \cup \phi'\) (\(\phi\) until \(\phi'\)), and \(\phi \mathcal{R} \phi'\) (\(\phi\) releases \(\phi'\)), similarly to the restriction of ATL\textsuperscript{*} to ATL in [Alur et al., 2002]. The result, Strategy Logic with simple goals (SL[SG]), can also be seen as the extension of ATL to arbitrary quantification on the agents’ strategies. We use SL[SG] to capture Stackelberg equilibria in games, express coercion-resistance in voting protocols [Tabatabaei et al., 2016], and characterize module checking for simple goals [Jamroga and Murano, 2014]. These are all relevant agent-related concepts, that cannot be expressed in ATL. Hence, SL[SG] is provably more expressive than ATL. Most importantly, we show that reasoning about SL[SG] is no more complex as ATL, as it also enjoys polynomial-time model checking. This is achieved by generalising the fixed-point procedures for ATL model checking. In consequence, Strategy Logic with simple goals offers an arguably better balance between expressivity and complexity than ATL. We further show the advantages by means of a case study, based on a simple voting scenario.
2 Logics for Strategies

We begin by recalling Strategy Logic (SL) [Mogavero et al., 2014], as well as its relevant syntactic fragments. Then, we provide an interpretation to these languages by means of concurrent game structures (CGS), as it is customary.

**Strategy Logic.** Fix an infinite set $\mathcal{AP}$ of atomic propositions (atoms), a finite set $\mathcal{Ag}$ of agents, and an infinite set $\mathcal{Var}$ of variables $x_0, x_1, \ldots$ for strategies. Formulas in Strategy Logic are defined as follows, for $p \in \mathcal{AP}$, $x \in \mathcal{Var}$, $a \in \mathcal{Ag}$:

$$\varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid X \varphi \mid \varphi U \varphi \mid \exists x \varphi \mid (x,a)\varphi$$

Next $X$ and until $U$ are temporal operators. The strategy quantifier $\exists x$ reads as “for some strategy $x$, …”, and the binding operator $(x,a)$ intuitively means that “by using strategy $x$, agent $a$ can achieve …”. Hereafter we use standard abbreviations, e.g., $\forall x \varphi$ for $\neg \exists x \neg \varphi$, as well as temporal operators releases $R$, globally $G$, and eventually $F$. We assume the standard definition of set free$(\varphi)$ of free agents and variables appearing in a formula $\varphi$ [Mogavero et al., 2014]. Moreover, shr$(x, \varphi)$ denotes the set of agents that use strategy $x$ in the evaluation of formula $\varphi$.

**Models.** Given sets $\mathcal{Ag}$ of agents and $\mathcal{AP}$ of atoms, a concurrent game structure (CGS) is a tuple $G = (S, s_0, (Act_a)_{a \in \mathcal{Ag}}, \tau, L)$ such that: $S$ is a non-empty finite set of states and $s_0 \in S$ is the initial state of $G$; for every agent $a \in \mathcal{Ag}$, $Act_a$ is a finite non-empty set of actions, and $ACT = \prod_{a \in \mathcal{Ag}} Act_a$ is the set of joint actions; $\tau: S \times ACT \rightarrow S$ is the transition function; finally, $L : S \rightarrow 2^{\mathcal{AP}}$ is the labelling. A path $h$ is a (finite or infinite) sequence $\pi \in S^* \cup S^\omega$ such that for every $j \geq 1$, $\pi_{j+1} = \tau(\pi_j, a_j)$ for some joint action $a_j \in ACT$. We distinguish between finite paths, or histories, and infinite paths, or computations. For a path $\pi$ and $j \geq 1$, $\pi_{\leq j}$ denotes the initial history of length $j$, and last$(h)$ is last element in history $h$.

**Strategies.** A memoryfull strategy for an agent $a \in \mathcal{Ag}$, or $a$-strategy, is a function $\sigma: S^+ \rightarrow Act_a$. The set of all strategies, for all agents, is denoted as $\Sigma(G)$. A joint strategy $\sigma_{\mathcal{Ag}}$ assigns a strategy for $a$ to every agent $a \in \mathcal{Ag}$. By the above definitions, given a strategy $\sigma$ for an agent $a$, if a different agent $b$ is such that the range of strategy $\sigma$ is a subset of $Act_b$ (i.e., $ran(\sigma) \subseteq Act_b$), then intuitively also agent $b$ can use strategy $\sigma$. Finally, an assignment is a function $\chi : \mathcal{Var} \cup \mathcal{Ag} \rightarrow \Sigma(G)$ such that for every agent $a \in \mathcal{Ag}$, $\chi(a)$ is a strategy for $a$. For $z \in \mathcal{Var} \cup \mathcal{Ag}$ and $\sigma \in \Sigma(G)$, the variant $\chi^z(\sigma)$ is the assignment that maps $z$ to $\sigma$ and coincides with $\chi$ on all other variables and agents. Given a history $h \in S^+$, an assignment $\chi$ defines a unique computation $\lambda(h, \chi) = h\chi^x(h)_{x \in L(h)}$, starting with $h$ and consistent with $\chi$.

**Semantics.** Given a CGS $G$, we inductively define the satisfaction relation $(G, h, \chi) \models \varphi$ where $h$ is a history, $\varphi$ is a formula, and $\chi$ is an assignment such that for every $x \in \mathcal{Var}$, $\chi(x)$ is a strategy for all agents in shr$(x, \varphi)$ (we omit the standard clauses for Boolean operators):

$$(G, h, \chi) \models p \iff p \in L(last(h))$$

$$(G, h, \chi) \models \Box \varphi \iff (G, \lambda(h, \chi)_{\leq |h|+1}, \chi) \models \varphi$$

We write $(G, h) \models \varphi$ iff $(G, h, \chi) \models \varphi$ for every assignment $\chi$, and $G \models \varphi$ iff $(G, s_0) \models \varphi$.

**Relevant Fragments.** One-Goal Strategy Logic (SL1G) is the syntactic fragment of SL where each temporal subformula is preceded by a binding prefix that mentions all agents in $\mathcal{Ag}$, and by a quantification prefix\(^1\) referring to all the preceding strategy variables [Mogavero et al., 2012]. Furthermore, ATL* can be seen as the fragment of SL1G that admits at most one alternation of strategic quantifiers (either $\exists \forall$ or $\forall \exists$), and does not allow to bind different agents to the same variable. Finally, ATL is the fragment of ATL* where each group of strategic quantifiers is followed by exactly one temporal operator. Interestingly, SL1G has been proved strictly more expressive than ATL*, while having the same complexity of the model checking problem [Mogavero et al., 2014]. In this paper, we look for an analogous extension of ATL.

**Example 1 (Simple Voting)** Consider a simple model of voting, inspired by [Jamroga et al., 2017a]. There are $k$ voters $v_1, \ldots, v_k$ and a single coercer $c$. At the beginning, each voter decides to wait or cast her vote for one of the $n$ candidates. Then, she can wait again, give her vote receipt to the coercer or refuse to give it. Finally, the coercer can either punish the voter, or refrain from punishment. The interaction between the coercer and different voters is independent. The CGS SV1\(_k\), for $k = 1$ and $n = 2$ is presented in Figure 1.

An example formula that holds in SV2,2 is $\forall x_1 \exists x_2 (x_1, c)(x_1, v_1)(x_2, v_2) F(voted_{v_1,1} \land \neg pun)$, expressing that, for every strategy of $c$, the first voter has a counterstrategy ensuring that she can vote for candidate 1 without getting punished, regardless of the other voter. The counterstrategy is simply to vote for 1, and then execute wait.

\(^1\)We will formally introduce both notions in the next section.
3 Strategy Logic with Simple Goals

Inspired by the relationship between the Alternating-time Temporal Logics ATL* and ATL, we introduce a novel restriction of SL[1G] to “simple” goals.

3.1 The Formal Language

We begin with some terminology. A binding prefix over sets \( A \subseteq Ag \) of agents and \( V \subseteq Var \) of variables is a finite sequence \( b \in \{ (a, x) \mid a \in A \text{ and } x \in V \}^{\mid A \mid} \) of length \( |b| = |A| \), such that every agent \( a \in A \) occurs exactly once in \( b \). In contrast, the same variable \( x \in V \) can occur several times in \( b \), i.e., intuitively, the same strategy denoted by \( x \) can be used by several agents in \( A \). A quantification prefix over a set \( V \subseteq Var \) of variables is a finite sequence \( \varphi \in \{ \exists x, \forall x \mid x \in V \}^{\mid V \mid} \) of length \( |\varphi| = |V| \), such that every variable \( x \in V \) occurs exactly once in \( \varphi \). Then, \( \text{Qnt}(V) \subset \{ (x, a) \mid a \in A \text{ and } x \in Var \}^{\mid A \mid} \) denote the sets of all quantification and binding prefixes over variables in \( V \) and \( A \).

Definition 2 (SL[SG]) The formulas in Strategy Logic with simple goals are defined in BNF as follows, where \( b \in \text{Bnd}(Ag) \), \( \varphi \in \text{Qnt}(\text{free}(\varphi)) \):

\[
\varphi ::= p | \neg \varphi | \varphi \land \varphi | \varphi \exists \varphi | \varphi \forall \varphi (b \varphi \varphi)
\]

By Def. 2, SL[SG] restricts SL[1G] to simple ATL objectives of type \( \exists \varphi, \forall \varphi \), and \( \varphi \land \varphi \land \varphi \) (the latter can be introduced by using U and dual quantification). We also consider the fragment SL\(_{-}[SG]\), for which Def. 2 is restricted to binding prefixes \( b \) where every variable occurs at most once. That is, one cannot bind different agents with the same strategy. Further, we define fragments SL\(_{n}[SG]\), for which quantification prefixes \( \varphi \) are restricted to at most \( n \) alternations of existential and universal quantifiers. Finally, we introduce fragments SL\(_{n}[SG]\) as the intersection of SL\(_{-}[SG]\) and each SL\(_{n}[SG]\).

Example 3 Clearly, the formula in Example 1 is a formula of SL[SG], more precisely of SL\(_{-}[SG]\).

In Fig. 2 we summarize the main syntactic inclusions between our fragments. We also remark that ATL can be thought of as the fragment of SL[SG] in which the clauses for formulas of type \( \varphi \exists \varphi \) and \( \varphi \forall \varphi \) are restricted to quantification prefixes \( \varphi \) with at most one alternation, as well as binding prefixes \( b \) where every variable occurs at most once. In fact, ATL corresponds exactly to SL\(_{-}[SG]\), whereas Computation Tree Logic (CTL) corresponds to SL\(_{0}[SG]\).

3.2 Expressiveness

An important observation that formally justifies our study is that SL[SG] is strictly more expressive than ATL. That is, SL[SG] allows to characterize all the properties expressible in ATL but not vice versa. We also show that SL[SG], SL\(_{-}[SG]\), and SL\(_{1}[SG]\) all differ with respect to their expressiveness.

Consider two logical systems \( L_1 \) and \( L_2 \). \( L_1 \) is at least as expressive as \( L_2 \) (written \( L_2 \subseteq L_1 \)) if every formula of \( L_2 \) can be equivalently translated to some formula of \( L_1 \). Moreover, \( L_1 \) is at least as distinguishing as \( L_2 \) (\( L_2 \preceq_d L_1 \)) if every pair of models that can be distinguished by a formula of \( L_2 \) can also be distinguished by some formula of \( L_1 \).

Theorem 4 SL\(_{-}[SG]\) (and thus also SL[SG]) has strictly greater expressive and distinguishing power than ATL.

Proof. The embedding of ATL into SL\(_{-}[SG]\) is straightforward. To show that ATL does not cover the distinguishing power (and hence also the expressive power) of SL\(_{-}[SG]\), we can use the counterexample from the proof of [Mogavero et al., 2014, Theorem 4.3].

![Figure 2: Syntactic inclusions for fragments of SL](image)

![Figure 3: Models M1 and M2 in the proof of Theorem 5](image)

Interestingly, we also have that for every \( n \in \mathbb{N} \), SL\(_{n}[SG]\) is strictly more expressive than SL\(_{n}[SG]\). That is, being able to bind the same strategy to different agents strictly increases the expressive power of Strategy Logic with simple goals.

Theorem 5 For every \( n \geq 0 \), we have:
1. SL\(_{n}[SG]\) \( \preceq_c SL_{n+1}[SG] \), and therefore SL\(_{n+1}[SG]\) \( \preceq_d SL_{n+1}[SG] \);
2. SL\(_{-}[SG]\) \( \preceq_c SL_{n}[SG] \), and therefore SL\(_{-}[SG]\) \( \preceq_d SL_{n}[SG] \).

Proof. We adapt the proof of [Belardinelli et al., 2012, Lemma 32.1]. Clearly, SL\(_{n}[SG]\) \( \preceq_c SL_{n+1}[SG] \) and therefore also SL\(_{n}[SG]\) \( \preceq_d SL_{n}[SG] \). Since SL\(_{0}[SG]\) \( \preceq_d SL_{0}[SG] \) and SL\(_{n}[SG]\) \( \preceq_d SL_{n}[SG] \), to show that SL\(_{n}[SG]\) \( \preceq_d SL_{n}[SG] \), we prove that:

\[
\text{SL}_{n}[SG] \not\preceq_d \text{SL}_{n+1}[SG]
\]

To do so, we provide two models that satisfy the same formulas in SL\(_{-}[SG]\) but are distinguished by a formula of SL\(_{n}[SG]\). Consider the CGS’s \( M_1 \) and \( M_2 \) with \( Ag = \{ 1, 2 \} \), depicted in Fig. 3. Since SL\(_{-}[SG]\) can be shown invariant under renaming of action labels, both models satisfy the same formulas of SL\(_{-}[SG]\). However, \( M_1 \) satisfies the SL\(_{0}[SG]\)-formula \( \exists x(x, 1)(x, 2) X p \), while \( M_2 \) does not. Thus, SL\(_{0}[SG]\) \( \not\preceq_d \text{SL}_{-}[SG] \) and therefore also SL\(_{n}[SG]\) \( \not\preceq_d \text{SL}_{n}[SG] \) and SL\(_{n}[SG]\) \( \not\preceq_c \text{SL}_{n+1}[SG] \) for every \( n \in \mathbb{N} \).
the voter (\(v\)). This specification can intuitively be read as: for every public strategy of the coercer (\(c\)), there exists a response by the voter (\(x_v\)) such that, no matter what the coercer does privately (\(x_c\)), the decoy process \(d\) can use decoy votes to make sure that the voter votes as she likes without being punished.

### 3.3 Motivating Examples and Applications

Strategy Logic enables to address complex interaction between strategies of different players, who may pursue adversarial, collaborative, or uncorrelated goals. Also, the goals can be based on sophisticated temporal patterns. This expressivity comes at the expense of tractability, or even decidability of decision problems for SL. At the other end, we have ATL where only a single alternation is allowed in strategy quantification. Moreover, the agents’ goals can only be phrased in terms of reachability. This results in a somewhat rigid specification language, albeit one with a big advantage: the model checking problem becomes tractable.

With SL[\(SG\)], we extend ATL by allowing arbitrarily many alternations of strategy quantifiers. We argue that the extra expressivity is useful, i.e., it allows for specification of relevant properties of agent interaction in multi-agent systems. To this end, we propose a number of motivating examples.

#### Example 6 (Stackelberg equilibria)

Stackelberg games concern scenarios where one player (the leader) exposes his strategy first, and the other player (the follower) adapts to that strategy by choosing her best response. For example, coercion in voting often has a Stackelberg structure: the coercer must lay out his coercion strategy in a believable way to force the voter to vote as requested. A desirable property in voting systems is that the voter can resist coercion, i.e., for every strategy of the coercer (\(c\)) there exists a response of the voter (\(v\)) such that for every run of the environment (\(e\)) the voter will have voted as she intended without being punished [Tabatabaei et al., 2016]. This is captured by the following formula of SL[\(SG\)]:

\[
\forall x_e \exists x_v \forall x_e \ (x_e, c)(x_e, v)(x_e, e) F(\text{voted}_{x_v} \land \neg \text{punished}).
\]

#### Example 7 (Coercion in voting, ctld.)

A more realistic model of coercion allows the coercer to split his strategy into a public and a private part. We can simulate that by splitting the model of the coercer into two agents: \(\text{cpub}\) and \(\text{cprv}\), each responsible for different parts of the coercion strategy. Moreover, some voting protocols include a process that can add decoy votes after the election to deceive the coercer. This can be incorporated into our specification of coercion-resistance as follows:

\[
\forall x_{x_1} \exists x_e \forall x_{x_2} \exists x_d \forall x_e \ (x_{x_1}, \text{cpub})(x_e, v)(x_{x_2}, \text{cprv}) (x_d, e) F(\text{voted}_{x_v} \land \neg \text{punished}).
\]

This specification can intuitively be read as: for every public strategy of the coercer (\(x_{x_1}\)), there exists a response by the voter (\(x_e\)) such that, no matter what the coercer does privately (\(x_{x_2}\)), the decoy process \(d\) can use decoy votes to make sure that the voter votes as she likes without being punished.

Example 8 (Module checking for strategic abilities)

The problem of module checking asks, for an open system embedded in a nondeterministic environment, whether the correctness specification holds for every possible strategy of the environment. It has been proved that module checking of either temporal or strategic specifications cannot be expressed in ATL [Jamroga and Murano, 2014]. The ability of an autonomous taxi (1) to serve an unbounded stream of customers (\(\epsilon\)) so that no accident occurs, regardless of what the agents \(a_1, \ldots, a_k\) do, can be captured as

\[
\forall x_e \exists x_1 \forall x_{a_1} \ldots \forall x_{a_k} \ (x_e, e)(x_{a_1}, t)(x_{a_1}, a_1) \ldots (x_{a_k}, a_k) \ G \neg \text{crash}.
\]

A careful reader may notice that the results in Section 3.2 indeed apply to the above specifications. That is, the formulas in Examples 6–8 cannot be equivalently expressed in ATL.

### 4 Model Checking

Given a CGS \(G\) and a formula \(\phi\) in language L, the model checking problem w.r.t. L consists in determining whether \(G \models \phi\). In this section we study model checking for Strategy Logic with simple goals and its fragments. These results are essential for applications of SL[\(SG\)] to the verification of multi-agent systems, including the specifications in Examples 6–8. Our main theoretical result is that verification of SL[\(SG\)] is tractable with respect to the size of the CGS (i.e., the number of its transitions) and the length of the formula.

We first prove that the fixed-point characterisation of ATL carry over to SL[\(SG\)].

**Proposition 9** The following formula is a validity in SL[\(SG\)]:

\[
\psi(\phi(\bigvee_{1} \phi(\bigvee_{2} \text{F}(\phi(\bigvee_{1} \phi(\bigvee_{2}))))
\]

**Proof.** The proof makes use of the preimage function \(\text{Pre}(\cdot)\) that is defined in Algorithm 2 and will be described in detail in the proof of Theorem 10. As regards the implication from left to right, suppose that \(\psi(\phi(\bigvee_{1} \phi(\bigvee_{2}))))\) holds in some state \(s\). Then, if \(\phi(\bigvee_{2})\) does not hold in \(s\), \(\phi(\bigvee_{1})\) does. Moreover, consider set \(Y \subseteq S\) such that \(s \in \text{Pre}(\phi(\bigvee_{1} \phi(\bigvee_{2}))))\) and \(\phi(\bigvee_{2})\) does not hold in \(Y\), for every \(s' \in Y\), we have \((G, s') = \phi(\bigvee_{1} \phi(\bigvee_{2}))\): indeed every strategy \(\sigma_{i}\) witnessing an existential quantifier \(Q_{i}\) in \(\psi\) at \(s\), also witnesses \(Q_{i}\) in history \(s'\). In particular, \((G, s') = \phi(\bigvee_{1} \phi(\bigvee_{2})))\) and, finally, \((G, s') = \phi(\bigvee_{1} \phi(\bigvee_{2})))\) by the way \(s'\) was chosen.

As for the implication from right to left, if \(\phi(\bigvee_{2})\) holds at state \(s\), then \(\phi(\bigvee_{1} \phi(\bigvee_{2})))\) is also the case. On the other hand, suppose that \((G, s') = \phi(\bigvee_{1} \phi(\bigvee_{2})))\) and \((G, s') = \phi(\bigvee_{2})\). In particular, every existential quantifier \(Q_{i}\) in \(\psi\) is witnessed in \(s\) by some strategy \(\sigma_{i}\), and in every “successor” state \(s' \in Y\), \(Q_{i}\) is witnessed in history \(s'\) by some (possibly different) strategy \(\sigma'_{i}\). Then, define strategies \(\sigma''_{i}(\cdot) = \sigma_{i}(\cdot)\) and \(\sigma''_{i}(s \cdot h) = \sigma'_{i}(h)\) for all \(h \in S^{+}\). In particular, each \(\sigma''_{i}\) witnesses \(Q_{i}\) at \(s\), that is, \((G, s) = \phi(\bigvee_{1} \phi(\bigvee_{2})))\). Observe the essential use of perfect recall in the construction of strategy \(\sigma''_{i}\) from \(\sigma_{i}\) and \(\sigma'_{i}\).

\(^{2}\)Admittedly, to represent module checking properly we need CGSs with non-deterministic transitions. Nonetheless, the characterisation of module checking in SL[\(SG\)] would remain the same.
By using Prop. 9 we can prove the main theoretical result.

**Theorem 10** Model checking $\mathsf{SL}[\mathsf{SG}]$ is $\mathsf{P}$-complete.

**Proof.** Recall that CTL corresponds to the $\mathsf{SL}_0[\mathsf{SG}]$ fragment of $\mathsf{SL}[\mathsf{SG}]$. Then, the lower bound follows immediately from the $\mathsf{P}$-hardness of model checking CTL [Katoen, 2008].

As for the upper bound, Algorithm 1 shows a procedure for model checking $\mathsf{SL}[\mathsf{SG}]$, which manipulates sets of states in $S$. The procedure is inspired by the standard model checking algorithm for ATL [Alur et al., 2002], but the preimage operator for the ATL modality \langle A \rangle X is now replaced by the $\mathsf{Pre}$ operator in Algorithm 2 for an arbitrary block of strategy quantifiers. Specifically, Algorithm 1 uses the following primitive operations:

- The function $\mathsf{Sub}$ returns a sequence of syntactic subformulas of a given formula $\varphi$.
- The function $\mathsf{Reg}$ returns the set $\{s \in S \mid p \in L(s)\}$ of states $s \in S$ labelled with a given atom $p \in AP$.
- The function $\mathsf{Pre}$, when given a quantification prefix $\varphi = Q_1x_1 \ldots Q_kx_k$, a binding prefix $b = (x_1, a_{1,1}) \ldots (x_1, a_{1,m_1}) \ldots (x_k, a_{k,1}) \ldots (x_k, a_{k,m_k})^3$, a set $Y \subseteq S$ of states, and a tuple $\vec{\alpha}$ of actions for all agents $a \in Ag$ such that $(x, a)$ appears in $b$ and $x$ does not appear in $\varphi$, returns the set of states $s \in S$ from which there exist transitions consistent with $\varphi$, $b$, and $\vec{\alpha}$, ending up in $Y$. Formally, $\mathsf{Pre}$ is a recursive function defined on the length of $\varphi$. For the base case $\varphi = \epsilon$, notice that $\alpha$ is a joint action for all agents in $Ag$, and therefore $\mathsf{Pre}$ returns the set of states $s \in S$ such that $\tau(s, \vec{\alpha}) \in Y$. As for the recursive step, $s \in \mathsf{Pre}(\varphi, b, Y, \vec{\alpha})$ iff for $\psi_1 = \exists \gamma$ (resp. $\forall$), for some (resp. every) action $\beta$, the case that $s \in \mathsf{Pre}(\psi_1, b, Y, \varphi_2, \vec{\alpha}, \beta)$, where function $\mathsf{update}(b, \vec{\alpha}, x, \beta)$, given a binding $b$, a tuple $\vec{\alpha}$ of actions, variable $x$, and action $\beta$, returns a tuple $\vec{\alpha}^\prime$ of actions such that for every $a \in Ag$, if $(x, a)$ appears in $b$ then $\alpha_{a,1} = \beta$, otherwise $\alpha_{a,1} = \alpha_a$. For more details on function $\mathsf{Pre}$, see Algorithm 2.

- Union, intersection, exclusion, and inclusion test for sets of states.

---

1The parameter $m_i$ represents the number of agents assigned to variable $x_i$, for each $1 \leq i \leq k$.

Algorithm 1 $\mathsf{SL}[\mathsf{SG}]$ Model Checking

```
procedure $\mathsf{MODEL-CHECKING}(G, \varphi)$
for all $\varphi'$ in $\mathsf{SL}(\varphi)$ do
  case $\varphi' = p$: [$\varphi' := \mathsf{Reg}(p)$;]
  case $\varphi' = \neg \varphi$: [$\varphi' := S \setminus \{\varphi\}$;]
  case $\varphi' = \theta_1 \land \theta_2$: [$\varphi' := \{\theta_1\} \cap \{\theta_2\}$;]
  case $\varphi' = \psi \land \theta$: [$\varphi' := \mathsf{Pre}(\psi, \theta, \{\theta\})$;]
  case $\varphi' = \psi \lor \theta$: [$\varphi' := \mathsf{Pre}(\psi, \theta, \{\theta\})$;]
  case $\varphi' = \psi \land \bar{\theta}$: [$\varphi' := \mathsf{Pre}(\psi, \theta, \{\theta\})$;]
  case $\varphi' = \psi \lor \bar{\theta}$: [$\varphi' := \mathsf{Pre}(\psi, \theta, \{\theta\})$;]
  case $\psi$: [$\varphi' := \psi(\beta, U \theta_2)$;]
  case $\varphi': Y := \emptyset; Z := \{\varphi_2\}$;
  while $Z \not\subseteq Y$ do
    $Y := Y \cup Z$;
    $Z := \mathsf{Pre}(\varphi, \theta, \beta, \epsilon, Y)$;
  end while
  [$\varphi' := \varphi_2 \cap \{\theta_1\}$;]
end for
end procedure
```

Algorithm 2 Preimage of a set $Y$ of states

```
procedure $\mathsf{PRE}(\varphi, b, Y, \vec{\alpha})$
  if $\varphi = \epsilon$ then return $(s \in S \mid \tau(s, \vec{\alpha}) \in Y)$;
  else if $\psi_2 = \exists \gamma$ then return
    $\bigcup_{\beta \in \mathcal{N}(\psi_1, \gamma)} \mathsf{Act}(\psi_1, \beta, Y, \mathsf{pre}(\varphi_1, b, Y, \mathsf{update}(b, \vec{\alpha}, \varphi_1, \beta)))$;
  else return
    $\bigcap_{\beta \in \mathcal{N}(\psi_1, \gamma)} \mathsf{Act}(\psi_1, b, Y, \mathsf{update}(b, \vec{\alpha}, \varphi_1, \beta)))$;
  end if
end procedure
```

Algorithm 1 works bottom-up on the structure of the formula; the cases of interest are for strategy formulas. For $\varphi' = \psi \land \theta$, the procedure calls function $\mathsf{Pre}$ to compute the set of states that are “bound” to end up in satisfaction set $\{\theta\}$. As regard $\varphi' = \psi \land \bar{\theta}$, the procedure computes the least fixed-point of operator $F(Z) = [\{\theta_2\} \cup \{\{\theta_1\} \cap \psi \lor \bar{\theta} \lor \psi \lor \bar{\theta} \in \mathcal{N}(\psi_1, \gamma)\}$.

We observe that, since $F$ is monotone, such a fixed-point always exists. Further, Algorithm 1 runs in polynomial time in the size of the CGS and the size of the formula. Termination of Algorithm 1 is guaranteed, as the state space $S$ is finite. Soundness and completeness can be proved by induction on the structure of the input formula $\varphi$ by using Proposition 9 for the case $\varphi' = \psi(\theta_1 \cup \theta_2)$. □

By Theorem 10 we immediately obtain that for all fragments of $\mathsf{SL}[\mathsf{SG}]$ in Fig. 2 model checking is $\mathsf{P}$-complete.

### 5 Experimental Evaluation

With $\mathsf{SL}[\mathsf{SG}]$, we propose a logic that enhances the expressivity of ATL while enjoying the same, polynomial-time complexity of model checking. Alternatively, we sacrifice some expressiveness of $\mathsf{SL}[1G]$ in order to reduce the verification complexity from highly tractable ($\mathsf{2EXPTIME}$-complete) to tractable. In consequence, one would expect that the verification algorithm for $\mathsf{SL}[\mathsf{SG}]$, presented in Section 4, should perform better than the general algorithm that handles the whole “one-goal” fragment of SL [Cermák et al., 2015].

However, theoretical complexity results rely on worst case complexity. Thus, it can be that the practical performance of $\mathsf{SL}[1G]$ model checking is much better than the $\mathsf{2EXPTIME}$ classification suggests. In particular, one may ask if the specialized $\mathsf{SL}[\mathsf{SG}]$ algorithm of the previous section performs significantly better than the general algorithm in [Cermák et al., 2015] on the formulas of $\mathsf{SL}[\mathsf{SG}]$. To answer this question, we have conducted a series of experiments with a scalable benchmark, based on the simple voting and coercion scenario of Examples 1 and 6.

We consider models $\mathsf{ESV}_{k,n}$ (Extended Simple Voting with $k$ voters and $n$ candidates), constructed as follows. The initial state $q_0$ is handled by a new election authority $ea$, which decides whether to implement low or high anti-coercion protection. The former choice leads to a copy of the $\mathsf{SV}_{k,n}$ CGS. The latter proceeds to a modified copy of the same model, with proposition $\mathsf{pun}$, satisfied only in the states where the coercer has decided to punish the voter and the voter gave her voting receipt.

Notice, however, that the number of transitions in a CGS can be exponential w.r.t. the number of agents.
Moreover, we consider the formula:

$$\exists x_0 \forall x_c \exists x_v \forall x_{v_2} \ldots \forall x_{v_k} (x_0,c)(x_c,c)(x_v_1,v_1) \ldots (x_{v_k},v_k)$$

$$F(\text{finish} \land \text{voted}_{1,1} \land \neg \text{pun}_1)$$

expressing that the election authority can make sure that, for every strategy of the coercer, voter 1 has a strategy to eventually complete her participation in the election, having voted for candidate 1 without getting punished. Clearly, the formula is true in every model $ESV_{k,n}$ (the strategy for $e$ being high protection, and for the voter to never give away her vote). In the experiments, we have only used models with $n = 2$. Thus, the number of voters ($k$) was the sole scaling factor.

We implemented the model checking algorithm in Section 4 in Python 3, using explicit representation of states and transitions. The tool is available on-line\footnote{https://github.com/slsigcai19/StrategyLogicSimpleGoals}. For the general $SL[1G]$ algorithm, we used the MCMAS-SL[1G] extension of MCMAS \cite{Cermák2015} and the MCMAS-SLK extension. The experiments were conducted on an Intel Core i7-6700 CPU with dynamic clock speed of 2.60–3.50 GHz and 32 GB RAM, running under 64bit Windows 10. The timeout was set to 5 hours. The experimental results are presented in Tab. 1. All times are given in seconds. The first two columns describe the model configuration and its size. The next two columns show the model generation time and the verification time for our algorithm. Finally, the last column presents the performance of the general $SL[1G]$ tool.

The results show significant improvement in running time when using the specialized algorithm (by orders of magnitude). Moreover, our implementation was able to verify models of up to nearly 50,000 states, while the more general tool handled only 10 times smaller models.

### Table 1: Experimental results for the simple voting model

<table>
<thead>
<tr>
<th>#v</th>
<th>#states</th>
<th>SL[SG]</th>
<th>SL[1G]</th>
<th>SLK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>tgen</td>
<td>tverif</td>
<td>tgen+tv</td>
</tr>
<tr>
<td>1</td>
<td>29</td>
<td>0.001</td>
<td>0.002</td>
<td>0.42</td>
</tr>
<tr>
<td>2</td>
<td>395</td>
<td>0.02</td>
<td>0.17</td>
<td>27.76</td>
</tr>
<tr>
<td>3</td>
<td>5573</td>
<td>0.98</td>
<td>45.31</td>
<td>5247.65</td>
</tr>
<tr>
<td>4</td>
<td>79187</td>
<td>30.40</td>
<td>8502.12</td>
<td>timeout</td>
</tr>
<tr>
<td>5</td>
<td>1130669</td>
<td>805.38</td>
<td>timeout</td>
<td>timeout</td>
</tr>
</tbody>
</table>

Related Work. Logics for strategies are a powerful tool for strategic reasoning in MAS. Their major bottleneck is the high complexity of the related decision problems. To overcome this, two main lines of research have been followed recently. One deals with semantical restrictions, mainly by limiting the number of strategies to evaluate \cite{Chatterjee2014, Vester2013, Brihaye2009, Jamroga2017, Bruyère2013}; while the other deals with the syntax, e.g., by looking at “simpler” fragments \cite{Alur2004, Pauly2002, Mogavero2014}. In this paper, we focused on this second line. Among such fragments, one of the most studied is ATL, a proper sublanguage of ATL\footnote{https://github.com/slsigcai19/StrategyLogicSimpleGoals} \cite{Alur2002}. Recently, the syntactic restriction to simple goals behind the success of ATL has also been investigated in the context of more powerful logics for strategic reasoning. In \cite{Malvone2018} the authors consider the simple vanilla fragment of Graded SL[1G] ($GSL[1G]$), but for two agents only. Hence, their contribution is orthogonal to ours: it is more general, as they can count strategies, but also more restricted, since it only considers two agents. Further, in \cite{Laroussinie2015} the authors consider the ATL-like restriction of ATL\footnote{https://github.com/slsigcai19/StrategyLogicSimpleGoals} with strategy contexts. However, this restriction does not reduce the expressive power nor the complexity of the related decision problems.

Future Work. A natural direction for future work is to look for more expressive, yet tractable, formalisms. For instance, \cite{Bulling2010} introduced ATL\footnote{https://github.com/slsigcai19/StrategyLogicSimpleGoals} that sits between ATL and ATL\footnote{https://github.com/slsigcai19/StrategyLogicSimpleGoals}. This fragment includes only formulas where each temporal operator is followed by a state formula, and allows cooperation modalities to be followed by a Boolean combination of path formulas. Notably, model checking ATL\footnote{https://github.com/slsigcai19/StrategyLogicSimpleGoals} is PSPACE-complete. We envisage an extension of SL[SG] similar to ATL\footnote{https://github.com/slsigcai19/StrategyLogicSimpleGoals}, hopefully with a PSPACE-complete model checking problem too. We also plan to release our model checking prototype as a verification tool.

Acknowledgements
F. Belardinelli acknowledges the support of the ANR JCJC Project SVeDaS (ANR-16-CE40-0021). W. Jamroga and D. Kurpiewski acknowledge the support of the National Centre for Research and Development NCBiR, Poland, and the National Research Fund FNR, Luxembourg, under the PollLux/FNR-INTER project VoteVerif (POLLUX-IV/1/2016).
References


