ATL$^+$ With Strategy Interaction*

Chung-Hao Huang[1] Sven Schewe[2] Farn Wang[1,3] Fang Yu[4]

1: Graduate Institute of Electronic Engineering, National Taiwan University
2: Dept. of Computer Sciences, University of Liverpool
3: Dept. of Electrical Engineering, National Taiwan University
4: Dept. of Management Information Systems, National Chengchi University

June 19, 2014

Abstract

We propose an extension to *ATL*$^+$ (*alternating-time logic*), called *BSIL* (*basic strategy-interaction logic*), for the specification of collaboration among the strategies of agents in a multi-agent system. Via the introduction of new modal operators for hierarchical strategy collaboration, BSIL allows for the specifications of a system strategy that can cooperate with several strategies of the environment for different objectives. We argue that such properties are important in practice and show that such properties are not expressible in *ATL**, *GL* (*game logic*), and *AMC* (*alternating μ-calculus*). Specifically, we show that BSIL is strictly more expressive than ATL$^+$, but incomparable with ATL*, GL, and AMC in expressiveness. We show that a memoryful strategy is necessary for fulfilling a specification in BSIL. We also study the complexity of the model and satisfiability checking problems of ATL$^+$ and BSIL. We show that BSIL model-checking can be performed in polynomial space and establish a matching PSPACE lower bound for model-checking the syntactic fragment ATL$^+$ of BSIL. The model complexity, however, is much lower: BSIL model-checking can be performed in time quadratic in the model (and exponential in the formula), and is PâŤĂcomplete for fixed formulas. Finally, we show that BSIL (and hence ATL$^+$) satisfiability is 2-EXPTIME complete, where the lower bound is inherited from CTL$^+$. Thus, the additional expressive power of BSIL does not come to additional computational cost.

Keywords: games, concurrent, turn-based, multi-agent, strategy, logic, model-checking, expressiveness, realisability, satisfiability

1 Introduction

The specification and verification of open systems focuses on the design of system interfaces that allow for the fulfillment of some objectives of the users while enforce certain safety policies of the systems. The theoretical challenges in designing such systems have drawn the attention of researchers from the game theory community. From the game theoretical perspective, the design problem of such open systems can be modeled as multi-agent

*The work is partially supported by NSC (Grant NSC 97-2221-E-002-129-MY3, Taiwan, ROC), by the Research Center for Information Technology Innovation, Academia Sinica, Taiwan, ROC, and by the Engineering and Physical Science Research Council (EPSRC), grant EP/H046623/1, UK

games. Some agents represent the system, while other agents represent the environment (or the users). An agent may have several objectives that may conflict with the other agents' objectives. On one hand, if a user objective is not in conflict with the system's objectives, then the user should be allowed to achieve the objective. On the other hand, if a user objective conflicts with the system's, then the user objective should be forbidden from fulfillment. For example, in a banking system, a user is allowed to withdraw money from her/his bank account, check balances of her/his account, etc. The user may also have other objectives forbidden by the system, including withdrawing money from others' accounts, checking balances of others' accounts, etc. The system wins the game in an execution (or a *play* in the jargon of game theory) if on one hand, the user's objectives allowed by the system are indeed fulfilled and on the other hand, her/his disallowed objectives do not happen. The goal of system design is, from the game theoretical perspective, to design a computable strategy of the system that enforces all the system objectives and also allows the users to achieve those objectives consistent with the system objectives. Such a strategy is called a *winning strategy* for the system.

At the moment, there are various logic that express such properties of strategic power of agents, including *ATL* (*alternating-time logic*), *ATL**, *AMC* (*alternating μ-calculus*), *GL* (*game logic*) [2], and *SL* (*strategy logics*) [5, 6, 15], for the specification of open systems. Each language also comes with a verification algorithm that helps to decide whether or not a winning strategy for the system exists. There is, however, a gap between the industrial need for efficient algorithms (and solvers) and the available technology offered from previous research. Frankly speaking, none of those languages represents a proper combination of expressiveness for close interaction among agent strategies and efficiency for specification verification. ATL, ATL*, AMC, and GL [2] allow us to specify that some players together have a strategy to satisfy some fully temporalised objective: strategy quantifiers mark the start of a state formula. As exemplified below, this falls short of what the industry needs in specification.

Consider the example of a bank that need specify their information system embodied as a system security strategy and allowing a client to use a strategy to withdraw money, to use a strategy to deposit money, and to use a strategy to query for balance. Moreover, the same system strategy should forbid any illegal operation on the banking system. Specifically, the same system strategy must accommodate all strategies of the client for 'good behavior' (i.e, behavior in line with the specification), while blocking all strategies of the client that refer to undesired behavior, and thus preventing the client from damaging the system. We will show that *ATL ATL**, and *GL* [2] do not support such specifications. For example, it is not possible to specify in those languages that the system strategies used both in a withdrawal transaction and in a deposit transaction must be the same. Consequently, verification techniques for specifications in those languages cannot capture such real-world objectives for open systems.

To solve the expressiveness problem in the above example, strategy logics were proposed in [5, 6, 15] that allow for the flexible quantification of high-order strategy variables in logic formulas. However, their verification complexities are prohibitively high and hinder them from practical application. In retrospect, the strategy profiles in such strategy logics can be combined in unrestricted ways. For example, in [15], we can write down the following

artificial and hypothetical property.

$$\langle\langle X\rangle\rangle[\![Y]\!]\langle\langle Z\rangle\rangle\Box((((1,X)\Diamond p)\wedge\bigcirc(2,Y)\Box q)\rightarrow(((2,Z)\Diamond q)\wedge\neg(3,Z)\Box q))$$

Here $\langle\langle X\rangle\rangle$ and $\langle\langle Z\rangle\rangle$ declare the existence of strategies named X and Y respectively. $[\![Y]\!]$ is a universal quantification on a strategy named Y. Then operator $(1,X)$, $(2,Y)$, $(2,Z)$, and $(3,Z)$ respectively bind strategy X, Y, Z, and Z to agents $1, 2, 2$, and 3. As can be seen, the language is very free in style. According to the experiences in temporal logic development, usually proper restrictions in the modal operations can lead to a spectrum of sub-logics with different expressiveness and model-checking efficiency. The following are two examples:

- The validity problem of $\mathcal{L}$ (the first-order langauge of unary predicate symbols and the binary predicate symbol $\leq$) is non-elementary [17] while PTL, with the same expressiveness as $\mathcal{L}$, is only PSPACE-complete [16].
- Between CTL [1] and CTL* [8,9], there are many subclasses of CTL* with various balances between expressiveness and verification efficiency. Fair CTL [10], as a natural class between CTL and CTL*, is expressive enough for many practical specifications and still enjoys a polynomial time model-checking complexity. There are also other subclasses of CTL* with various balance considerations [4, 7, 9, 14].

As can be seen, subclasses of temporal logics with proper balance between expressiveness and verification efficiency are not only theoretically interesting and can also be practically useful. Indeed, most specifications in real-world projects come in simple structures, for example, safety, liveness, etc. Thus it would be interesting to see what "natural" subclasses of strategy logics can be identified with a proper balance between expressiveness and model-checking efficiency. Moreover, it would be practical and appealing if the subclass can be characterized with elegant syntax. Indeed it is the purpose of this manuscript to propose new natural modal operators for strategy collaborations and extend ATL for a subclass of strategy logic.

In the following, we use the classical prisoner's dilemma to explain how we can design new modal operators for structured strategy collaboration to achieve a balance between expressiveness and model-checking efficiency.

Example 1 **Prisoner's dilemma** Suppose the police is interrogating three suspects (prisoners). The police has very little evidence. A prisoner may cooperate (with his/her peers) and deny all charges made by the police. If all deny, they are all acquitted of all charges. However, each prisoner may choose to betray his/her peers and provide the police with evidence. If more than one prisoner choose to betray their peers, all will be sentenced and stay in jail. If only one chooses to betray, then the other prisoners will stay in jail, while (s)he will be a 'dirty witness' and all charges against him/her will be dropped.

We may want to specify that the three prisoners can cooperate with each other (by denying all charges), and will not be in jail. Let j_a be the proposition for prisoner a in jail. This can be expressed in Alur, Henzinger, and Kupferman's ATL*, GL, or AMC [2], respectively, as follows.[1]

[1] Note that the three example formulas are not equivalent.

ATL*: $\langle 1,2,3\rangle \bigwedge_{a\in[1,3]} \Diamond\neg j_a$
GL: $\exists\!\!\exists\{1,2,3\}.\bigwedge_{a\in[1,3]} \forall\Diamond\neg j_a$
AMC: $\mathbf{lfp}x.\langle 1,2,3\rangle \bigcirc \bigwedge_{a\in[1,3]}(x \vee \neg j_a)$

Here "$\langle 1,2,3\rangle$" and "$\exists\!\!\exists\{1,2,3\}$" are both existential quantifiers on the collaborative strategy among prisoners $1,2$, and 3. Such a quantifier is called a *strategy quantifier* (*SQ*) for convenience. Operator '**lfp**' is the least fixpoint operator. Even though we can specify strategies employed by sets of prisoners and there is a natural relationship (containment) between sets with such logics, there is no way to relate strategies to each other. For example, if prisoners 1 and 2 are really loyal to prisoner 3, they can both deny the charges, make sure that prisoner 3 will not be in jail, and let prisoner 3 to decide whether they will be in jail. ■

The research of strategies for related properties has a long tradition in game theory. It is easy to see the similarity and link between the specification problems for the prisoner's dilemma and the banking system. This observation suggests that finding a language with an appropriate and natural balance between the expressive power and the verification complexity of a specification language is a central challenge.

To meet this challenge, we propose an extension of ATL, called *BSIL* (*basic strategy-interaction logic*). In a first step, we extend ATL to ATL$^+$, where ATL$^+$ is the natural extension obtained by allowing for Boolean connectives of path quantifiers. (Cf. [4, 7] for the similar extension of CTL to CTL$^+$.) We then introduce a new modal operator called *strategy interaction quantifier* (*SIQ*). In the following, we use several examples in the prisoner's dilemma to explain BSIL, starting with the following specification for the property discussed at the end of Example 1.

$$\langle 1,2\rangle((\langle +\rangle\Diamond\neg j_3) \wedge (\langle +3\rangle\Diamond\neg(j_1 \vee j_2)) \wedge \langle +3\rangle\Box(j_1 \wedge j_2)) \tag{A}$$

Here "$\langle +3\rangle$" is an existential SIQ that reasons over strategies of Prisoner 3 for collaborating with the strategies of Prisoners 1 and 2 introduced by the parent SQ "$\langle 1,2\rangle$". Similarly, "$\langle +\rangle$" means that no collaboration of any prisoner is needed. (For conciseness, we omit "$\langle +\rangle$" sometimes.) We also call an SIQ an SQ. In BSIL formulas, we specifically require that no SIQ can appear as a topmost SQ in a path subformula.

As can be seen, SIQ imposes a hierarchical style of strategy collaboration which seems natural for practical specification. Consider another example. If Prisoner 1 really hates the others, (s)he can always betray other prisoners, making sure that Prisoners 2 and 3 will be in jail, and let them decide whether (s)he will be in jail, too. This property can be expressed in BSIL as follows.

$$\langle 1\rangle((\Box(j_2 \wedge j_3)) \wedge (\langle +2,3\rangle\Diamond\neg j_1) \wedge \langle +2,3\rangle\Box j_1) \tag{B}$$

One restriction of BSIL is that no negation between an SIQ and its parent SIQ or SQ is allowed. This restriction then also forbids universal SIQ. While at first glance, it seems less than elegant in mathematics, it is both necessary for verification complexity and compatible with practical specification styles. When we take a closer look, in fact, there is an implicit universal SIQ at the end of every maximal syntax path from an SQ to its descendant SIQ. Thus, BSIL allows for the specification of the ways in combining system strategy profiles that can be computed

statically to enforce the system policy against any hostile strategy profile of those agents not participating in the the system strategy profiles. If we explicitly allow for universal SIQs in BSIL, then the strategy profiles associated with existential SIQs in the scope of a universal SIQs may have to be computed dynamically. Thus the explicit universal SIQs will not only necessarily blow up the verification complexity, but also contradict the main-stream of game theory for statically calculable strategies. In contrast, some strategy logics [15] allow for the specification of strategy profiles that are not statically calculable.

In this work, we establish that BSIL is incomparable with ATL*, GL, and AMC in expressiveness. Although the strategy logics [5, 6, 15] are superclasses to BSIL with their flexible quantification of strategies and binding to strategy variables, their model-checking[2] complexity are all doubly exponential time hard. In contrast, BSIL enjoys a PSPACE-complete model-checking complexity for turn-based and concurrent game graphs. This may imply that BSIL could be a better balance between expressiveness and verification efficiency than ATL*, GL, AMC [2], and SL [5,15]. Further related work is the stochastic game logic (SGL) by Baier, Brázdil, Größer, and Kucera [3], which allows for expressing strategy interaction. However, for memoryful strategies, the model-checking problem of SGL is undecidable.

We also establish some other properties of BSIL. We show that the strategies for BSIL properties against turn-based games need to be memoryful. We prove that the BSIL model-checking problem is PSPACE-complete. However, the PSPACE model-checking algorithm need enumerate the labelings on computation trees and may suffer from high time complexity. We thus also present an alternative model-checking algorithm with time complexity quadratic in the size of a game graph and exponential only in the size of a BSIL specification. We also establish that the BSIL realisability problem is complete for doubly exponential time.

The article is organized as follows. Section 2 explains concurrent and turn-based game graphs for the description of multi-agent systems. Section 3 presents BSIL and ATL$^+$, and establishes the need of memoryful strategies. Section 4 discusses the expressiveness of BSIL. Section 5 establishes the PSPACE-completeness of model-checking problem. In Section 6, we present the alternative model-checking algorithm based on weak alternating automata construction. Section 7 discusses the complexity of the BSIL realisability problem.

2 Game graphs

2.1 Concurrent games

A *concurrent game* is played by many agents that make their moves concurrently. Such a game can formalized with the following definition.

Definition 1 *A concurrent game graph is a tuple* $A = \langle m, Q, r, P, \lambda, R, \Delta, \delta \rangle$ *with the following restrictions.*

[2] A model-checking problem is to check whether a given model (game graphs in this work) satisfies a logic formulas (in ATL and its extensions in this work).

- m *is the number of agents in the game.*
- Q *is a finite set of states.*
- $r \in Q$ *is the* initial state *of* A*.*
- P *is a finite set of atomic propositions.*
- *Function* $\lambda : Q \mapsto 2^P$ *labels each state in* Q *with a set of atomic propositions.*
- $R \subseteq Q \times Q$ *is the set of transitions.*
- Δ *is a set of tokens that can be issued by the agents during transitions.*
- $\delta : (R \times [1, m]) \mapsto \Delta$ *is a function that specifies the token (move symbol) issued by each agent in a transition.*

During a transition, each agent selects a token. If there is no transition matching all the tokens selected by the agents, then there is no transition. Otherwise, the matching transition takes place. ■

In Figure 1, we have the graphical representation of a concurrent game graph. The ovals represent states while

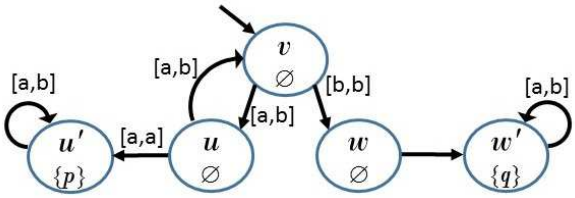


Figure 1: A concurrent game graph

the arcs represent state transitions. We also put down the λ values inside the corresponding states. On each edge, we label the tokens issued by the agents. Specifically, the label on arrow (q, q') is $[\delta((q, q'), 1), \ldots, \delta((q, q'), m)]$. For example, in Figure 1, on edge (v, u), we have label $[a, b]$ meaning that to make the transition, agent 1 has to choose token a while agent 2 has to choose b.

For convenience, in the remaining part of the manuscript, we assume that we are always in the context of a given game graph $\mathcal{G} = \langle m, Q, r, P, \lambda, R, \Delta, \delta \rangle$. Thus, when we write $Q, r, P, \lambda, R, \Delta$, and δ we respectively refer to the corresponding components of this $\mathcal{G}$.

A *state predicate* of P is a Boolean combination of elements in P. The satisfaction of a state predicate η at a state q, in symbols $q \models \eta$, is defined in the standard way.

A *play* is an infinite path in a game graph. A play is *initial* if it begins with the initial state. Given a play $\rho = \bar{q}_0 \bar{q}_1 \ldots$, for every $k \geq 0$, we let $\rho(k) = \bar{q}_k$. Also, given $h \leq k$, we let $\rho[h, k]$ denote $\rho(h) \ldots \rho(k)$ and $\rho[h, \infty)$ denote the infinite tail of ρ from $\rho(h)$. A *play prefix* is a finite segment of a play from the beginning of the play. Given a play prefix $\rho = \bar{q}_0 \bar{q}_1 \ldots \bar{q}_n$, we use $|\rho| = n + 1$ for the *length* of ρ. For convenience, we use *last*(ρ) to denote the last state in ρ, i.e., $\rho(|\rho| - 1)$.

Let Q^* be the set of finite sequences of states in Q. For an agent $a \in [1, m]$, a *strategy* σ for a is a function from Q^* to Δ. An *agency* A of $[1, m]$ is an integer subset of $[1, m]$. For example, "$\{1, 3, 4\}$" represents the agency that consists of agents 1, 3, and 4. A *strategy profile* (or *S-profile*) Σ of an agency $A \subseteq [1, m]$ is a partial function from $[1, m]$ to the set of strategies such that, for every $a \in [1, m]$, $a \in A$ iff $\Sigma(a)$ is defined. The composition of two S-profiles Σ, Π, in symbols $\Sigma \circ \Pi$, is defined with the following restrictions for every $a \in [1, m]$.

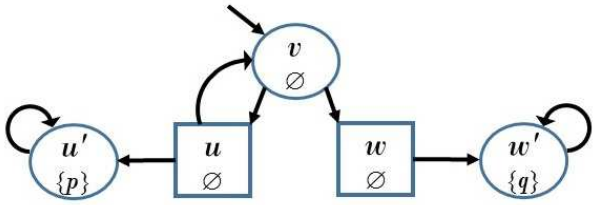
- If $\Pi(a)$ is defined, then $\Sigma \circ \Pi(a) = \Pi(a)$.
- If $\Sigma(a)$ is defined and $\Pi(a)$ is undefined, then $\Sigma \circ \Pi(a) = \Sigma(a)$.
- If $\Sigma(a)$ and $\Pi(a)$ are both undefined, then $\Sigma \circ \Pi(a)$ is also undefined.

We will use composition of S-profiles to model inheritance of strategy bindings from ancestor formulas.

A play ρ is compatible with a strategy σ of an agent $a \in [1, m]$ iff for every $k \in [0, \infty)$, $\delta((\rho(k), \rho(k+1)), a) = \sigma(\rho[0, k])$. The play is compatible with an S-profile Σ of agency A iff for every $a \in A$, the play is compatible with $\Sigma(a)$ of agent a.

2.2 Turn-based games

Another popular game structure is *turn-based game* in which at each state, at most one agent gets to decide the next state. For example, in figure 2, we have the graphical representation of a turn-based game graph with initial state v. The ovals and squares represent states respectively of agent 1 and agent 2. The arcs represent state transitions.



◯ belongs to Agent 1 and □ belongs to Agent 2.

Figure 2: A turn-based game graph

In fact, every turn-based game can be represented as a special case of concurrent games. Specifically, a turn-based game $\mathcal{G} = \langle m, Q, r, P, \lambda, E, \Delta, \delta \rangle$ can be viewed as a concurrent game with the following restrictions.

- $\Delta = Q \cup \{\perp\}$ where $\perp$ denotes a dummy move not in Q.
- For every $(q, q') \in R$, if q belongs to agent a, then $\delta((q, q'), a) = q'$ and for every $a' \neq a$, $\delta((q, q'), a') = \perp$.
- For every $(q_1, q_2), (q_3, q_4) \in R$ and agent a, $\delta((q_1, q_2), a) = \perp$ if and only if $\delta((q_3, q_4), a) = \perp$. This restriction says that every state can be owned by only one agent.

For convenience, for a turn-based game, the owner of a state q, $\omega(q)$ in symbols, is defined as agent a with $\forall (q, q') \in E(\delta((q, q'), a) = q'$. For ease of notation, we denote with $Q_a = \{q \in Q \mid \omega(q) = a\}$ the states owned by an agent

a.

In the investigation of many research issues, turn-based game graphs are easier to handle than concurrent game graphs. So in latter sections, we sometimes use turn-based game graphs in examples and explanation of the theory as we see fit.

3 Basic Strategy Interaction Logic (BSIL)

3.1 BSIL syntax

For concurrent game graph $\mathcal{G}$ of m agents, we have three types of formulas: *state formulas*, *tree formulas*, and *path formulas*. State formulas describe properties of states. Tree formulas describe interaction of strategies. Path formulas describe properties of plays. BSIL formulas are constructed with the following three syntax rules, respectively, for state formulas ϕ, tree formulas τ, and path formulas θ.

$$\begin{array}{rcl} \phi & ::= & p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \langle A\rangle\tau \mid \langle A\rangle\theta \\ \tau & ::= & \tau_1 \vee \tau_2 \mid \tau_1 \wedge \tau_2 \mid \langle +A\rangle\tau_1 \mid \langle +A\rangle\theta \\ \theta & ::= & \neg\theta_1 \mid \theta_1 \vee \theta_2 \mid \bigcirc \phi_1 \mid \phi_1 \mathrm{U} \phi_2 \end{array}$$

Here, p is an atomic proposition in P and A is an agency of $[1, m]$. $\langle A\rangle$ is a *strategy quantifier* (*SQ*) and $\langle +A\rangle$ is a *strategy interaction quantifier* (*SIQ*). $\langle A\rangle\psi$ means that there exists an S-profile for the agency A that makes all plays satisfy ψ. Formulas of the form $\langle +B\rangle\psi_1$ must happen within an SQ. Intuitively, they mean that there exists an S-profile of B that collaborates with the strategies declared with ancestor formulas to make ψ_1 true. For convenience, we view SQs as special cases of SIQs.

State formulas ϕ are called *BSIL formulas*. From now on, we assume that we are always in the context of a given BSIL formula χ. Note that we strictly require that strategy interaction cannot cross path modal operators. This restriction is important and allows us to analyze the interaction of strategies locally in a state and then enforce the interaction along all paths from this state.

For convenience, we also use the common shorthands.

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathit{true} & \equiv & p \vee (\neg p) & \mathit{false} & \equiv & \neg\mathit{true} \\ \psi_1 \wedge \psi_2 & \equiv & \neg((\neg\psi_1) \vee (\neg\psi_2)) & \psi_1 \Rightarrow \psi_2 & \equiv & (\neg\psi_1) \vee \psi_2 \\ \Diamond\phi_1 & \equiv & \mathit{true}\ \mathrm{U}\phi_1 & \Box\phi_1 & \equiv & \neg\Diamond\neg\phi_1 \end{array}$$

The SQs and SIQs introduced above are all existential in that they are satisfied with one S-profile. Note that there is no universal SQs and SIQs in BSIL. This is purely for the complexity of the model-checking algorithm (and problem) that we are going to present later. Thus BSIL can be used to specify the different ways of combining S-profiles to enforce system policy.

For ease of notations, we may abbreviate $\langle\{a_1, \ldots, a_n\}\rangle$ and $\langle +\{a_1, \ldots, a_n\}\rangle$ as $\langle a_1, \ldots, a_n\rangle$ and $\langle +a_1, \ldots, a_n\rangle$, respectively.

3.2 BSIL semantics

BSIL subformulas are interpreted with respect to S-profiles. A state or a tree formula ϕ is satisfied at a state q with S-profile Σ, denoted $\mathcal{G}, q \models_\Sigma \phi$, if, and only if, the following inductive constraints are satisfied.

- $\mathcal{G}, q \models_\Sigma p$ iff $p \in \lambda(q)$.
- For state formula ϕ_1, $\mathcal{G}, q \models_\Sigma \neg\phi_1$ iff $\mathcal{G}, q \models_\Sigma \phi_1$ is false.
- For state or tree formulas ψ_1 and ψ_2, $\mathcal{G}, q \models_\Sigma \psi_1 \wedge \psi_2$ iff $\mathcal{G}, q \models_\Sigma \psi_1$ and $\mathcal{G}, q \models_\Sigma \psi_2$.
- For state or tree formulas ψ_1 and ψ_2, $\mathcal{G}, q \models_\Sigma \psi_1 \vee \psi_2$ iff either $\mathcal{G}, q \models_\Sigma \psi_1$ or $\mathcal{G}, q \models_\Sigma \psi_2$.
- $\mathcal{G}, q \models_\Sigma \langle A\rangle\tau$ iff there exists an S-profile Π of A with $\mathcal{G}, q \models_\Pi \tau$.
- $\mathcal{G}, q \models_\Sigma \langle +A\rangle\tau$ iff there exists an S-profile Π of A with $\mathcal{G}, q \models_{\Sigma\circ\Pi} \tau$. Here, the composition $\Sigma \circ \Pi$ of the S-profiles Σ and Π models the inheritance of strategy bindings, Σ, from ancestor formulas.
- $\mathcal{G}, q \models_\Sigma \langle A\rangle\theta$ iff there exists an S-profile Π of A such that, for all plays ρ from q compatible with Π, $\rho \models_\Pi \theta$ holds. Intuitively, this means that ρ satisfies path formula θ with S-profile Π.
- $\mathcal{G}, q \models_\Sigma \langle +A\rangle\theta$ iff there exists an S-profile Π of A such that, for all plays ρ from q compatible with $\Sigma \circ \Pi$, $\rho \models_{\Sigma\circ\Pi} \theta$ holds.

A play ρ satisfies a path formula θ with S-profile Σ, in symbols $\rho \models_\Sigma \theta$, iff the following restrictions hold.

- For a path formula θ_1, $\rho \models_\Sigma \neg\theta_1$ iff it is not the case that $\rho \models_\Sigma \theta_1$.
- For path formulas θ_1 and θ_2, $\rho \models_\Sigma \theta_1 \vee \theta_2$ iff either $\rho \models_\Sigma \theta_1$ or $\rho \models_\Sigma \theta_2$.
- $\rho \models_\Sigma \bigcirc\psi_1$ iff $\mathcal{G}, \rho(1) \models_\Sigma \psi_1$.
- $\rho \models_\Sigma \psi_1 \mathrm{U} \psi_2$ iff there exists an $h \geq 0$ with $\mathcal{G}, \rho(h) \models_\Sigma \psi_2$ and for all $j \in [0, h)$, $\mathcal{G}, \rho(j) \models_\Sigma \psi_1$.

For convenience, we let $\perp$ be a null S-profile, i.e., a function that is undefined on everything. If ϕ_1 is a BSIL (state) formula and $\mathcal{G}, q \models_\perp \phi_1$, then we may simply write $\mathcal{G}, q \models \phi_1$. If $\mathcal{G}, r \models \phi_1$, then we also write $\mathcal{G} \models \phi_1$.

3.3 ATL$^+$

ATL$^+$ is the syntactic fragment of BSIL given by the following grammar

$$\begin{array}{rcl} \phi & ::= & p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \langle A\rangle\theta \\ \theta & ::= & \neg\theta_1 \mid \theta_1 \vee \theta_2 \mid \bigcirc\phi_1 \mid \phi_1 \mathrm{U} \phi_2 \end{array}$$

ATL$^+$ can also be viewed as an extension of ATL [2] that, similar to ...'s extension of CTL to CTL$^+$ [4, 9], allows for the Boolean combination of path formulas. As such, all complexities of ATL$^+$ must reside between those of BSIL and ATL as well as between those of BSIL and CTL$^+$, which we will use to establish the lower bounds for ATL$^+$ and BSIL.

3.4 Memory

In this subsection, we show a simple example that the agents need memory to achieve their objective for ATL$^+$ specifications. This is is exemplified by the simplest possible case: the turn based game in Figure 3 with one agent,

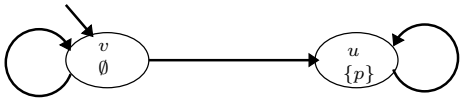


Figure 3: A simple turn-based game that demands memoryful strategies

two states, one atomic proposition, and two memoryless strategies that do not count the unreached states in the histories. For the ATL$^+$ sentence $\langle 1\rangle(\langle +\rangle\neg \bigcirc p) \wedge \langle +\rangle\Diamond p$, apparently agent 1 needs memory to enforce it.

Lemma 1 *The strategies of the agents in ATL$^+$ (and thus in BSIL) specifications need memory. This even holds for the single agent case.* ■

4 Expressive Power of BSIL

In this section, we establish that BSIL is incomparable with ATL*, AMC, and GL [2] in expressiveness.

4.1 Comparision with ATL*

It is easy to see that BSIL is a super-class of ATL. Thus we have the following lemma.

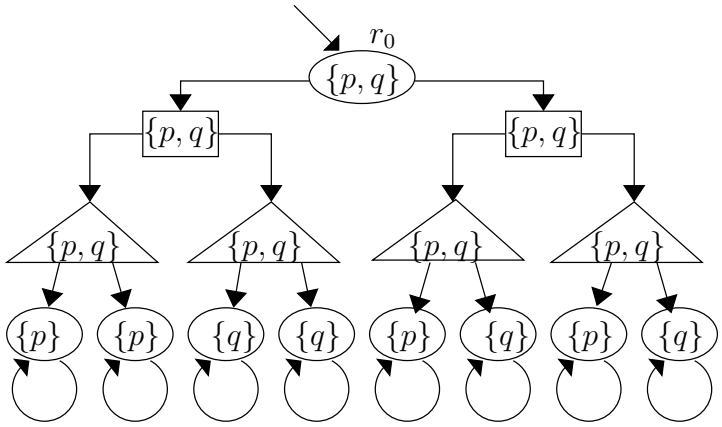
Lemma 2 *BSIL is at least as expressive as ATL.* ■

Lemmas 3 and 4 establish that ATL* and BSIL are incomparable.

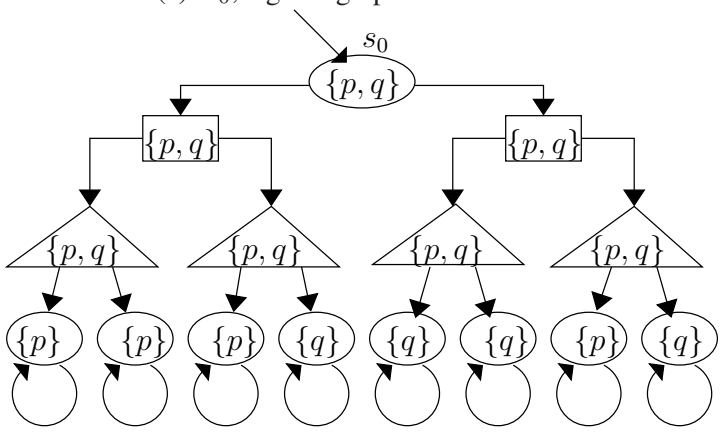
Lemma 3 *For every ATL* formula ϕ, there are two game graphs that ϕ cannot distinguish while $\langle 1\rangle((\langle +2\rangle\Box p) \wedge \langle +2\rangle\Box q)$ can.*

Proof : The proof is by an inductive construction of two families $G_0, \ldots, G_k, \ldots$ and $H_0, \ldots, H_k, \ldots$ of turn-based game graphs that no ATL* formula with k SQs can distinguish G_k and H_k. We prove the lemma by induction on k.
Base case: Assume that we have an ATL* formula ϕ with at most one SQ. Then we have the two game graphs in Figure 4 for three agents. Note that the two game graphs have the same set of traces. Thus they cannot be distinguished by path formulas. Then there are the following case analysis of ATL* formulas.

- **Case 1:** *ϕ is $\langle A\rangle\phi_1$ where ϕ_1 is a Boolean combination of path formulas.* Note that ϕ_1 characterizes a set of traces. For convenience of discussion, we call a trace that stabilizes to $\{p\}$ a p-trace. A trace that stabilizes to $\{q\}$ is called a q-trace. Also, for a path formula θ, we let $[\![\theta]\!]$ denote the set of traces characterized by θ. For the two game graphs in figure 4, there are only two types of traces. Thus there are only the following four different subsets of traces.
  - The set of no trace, denoted $\emptyset$.
  - The set of p-traces, denoted $\{\Box p\}$.



(a) G_0, a game graph for base case.



(b) H_0, another game graph for base case.
◯ belongs to Agent 1; □ belongs to Agent 2; and △ belongs to Agent 3.

Figure 4: Base cases for the expressiveness of BSIL over ATL*

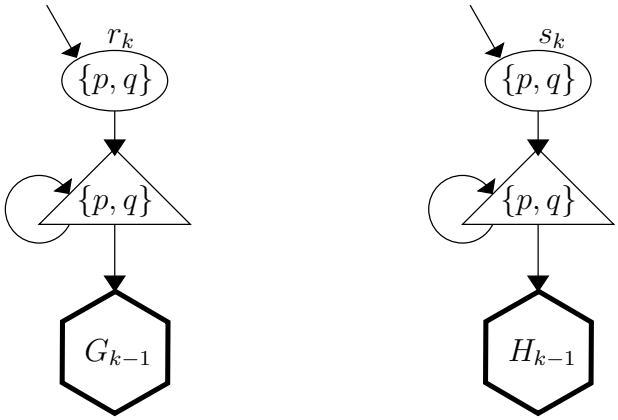
- The set of q-traces, denoted $\{\Box q\}$.
- The set of p-traces and q-traces, denoted $\{\Box p, \Box q\}$.

The following case analysis shows that, for every trace subset S and every agency A, there exists a strategy of A to characterize S in figure 4(a) iff there exists a strategy for A to characterize S in figure 4(b).

– **Case 1a:** *ϕ is* $\langle\emptyset\rangle\phi_1$. In this case, there is no strategy and the sets of traces imposed by no strategy in the two state graphs are both $\{\Box p, \Box q\}$. Thus, $\langle\emptyset\rangle\phi_1$ cannot distinguish the trace sets of the two state graphs.

– **Case 1b:** *ϕ is* $\langle 1\rangle\phi_1$. From figure 4, no matter what strategies agency $\{1\}$ may choose, the trace sets for the two game graphs are both $\{\Box p, \Box q\}$. Thus, $\langle 1\rangle\phi_1$ cannot distinguish the trace sets of the two state graphs.

– **Case 1c:** *ϕ is* $\langle 2\rangle\phi_1$. This case is similar to case 1b.



(a) G_k (b) H_k

$\bigcirc$ belongs to Agent 1; $\Box$ belongs to Agent 2; and $\triangle$ belongs to Agent 3.

Figure 5: Inductive cases for the expressiveness of BSIL over ATL*

- **Case 1d:** *ϕ is* $\langle 3\rangle\phi_1$. This case is similar to case 1b.
- **Case 1e:** *ϕ is* $\langle 1,2\rangle\phi_1$. Agency $\{1,2\}$ can cooperate to force three trace sets: $\{\Box p\}, \{\Box q\}, \{\Box p, \Box q\}$ in both of the game graphs in figure 4. Thus, $\langle 1,2\rangle\phi_1$ cannot distinguish the trace sets of the two state graphs.
- **Case 1f:** *ϕ is* $\langle 1,3\rangle\phi_1$. This case is similar to case 1e.
- **Case 1g:** *ϕ is* $\langle 2,3\rangle\phi_1$. This case is similar to case 1e.
- **Case 1h:** *ϕ is* $\langle 1,2,3\rangle\phi_1$. Agency $\{1,2,3\}$ can cooperate to force three trace sets: $\{\Box p\}, \{\Box q\}, \{\Box p, \Box q\}$ in both of the game graphs in figure 4. Thus, $\langle 1,2,3\rangle\phi_1$ cannot distinguish the trace sets of the two state graphs.

• **Case 2:** *ϕ is a Boolean combination of formulas in case 1.* Since case 1 does not distinguish the two game graphs, this case can neither do it.

Thus the base case is proven.

Induction step: We use structural induction on ATL* formulas to prove the step. If ϕ has k SQs, then we can construct the two graphs in figure 5. It is clear that $\langle 1\rangle((\langle +2\rangle\Box p)\wedge(\langle +2\rangle\Box q))$ can distinguish G_k and H_k. However, to tell the difference between the two graphs, we need a modal subformula of ϕ that can tell the difference of G_{k-1} and H_{k-1}. But according to the inductive hypothesis, this is impossible. Thus the lemma is proven. ■

Lemmas 2 and 3 together establish that ATL is strictly less expressive than BSIL.

Lemma 4 *ATL* formula $\langle 1\rangle\Box\Diamond p$ is not equivalent to any BSIL formula.*

Proof : The proof is similar to the proof for the inexpressibility of $\langle 1\rangle\Box\Diamond p$ with ATL [2]. ■

Lemmas 3 and 4 together establish that ATL* and BSIL are not comparable in expressiveness.

4.2 Comparision with GL

GL separates strategy quantificaitons from path quantifications. In comparison, ATL* and BSIL combines these two quantifications into SQs and SIQs. Thus with GL, we can specify that, for all S-profiles of A, there exists a play saitsfying ψ_1. The (existential) strategy quantification for agency A of GL is of the form $\exists A.\psi_1$. The path quantifications are just ordinary CTL modalities: $\forall\Box, \forall$U, $\exists\Box$, and $\exists$U.

The following two lemmas show the relation between GL and BSIL.

Lemma 5 *For every GL formula ϕ, there are two game graphs that ϕ cannot distinguish while $\langle 1\rangle((\langle +2\rangle\Box p) \wedge \langle +2\rangle\Box q)$ can.*

Proof : The proof is similar to the one for Lemma 3 and is by induction on the number of modal operators k.

Base case: Assume that ϕ has only one modal operator. Then we have the two game graphs in figure 4 for three agents. Then there are the following case analysis of GL formulas.

- **Case 1:** *ϕ is $\exists A.\phi_1$ where ϕ_1 is a Boolean combination of formulas of the form $\exists\psi$ or $\forall\psi$ where ψ is a path formula.* Note that ϕ_1 characterizes a set of states that either start a play satisfying ψ or start only plays satisfying ψ. The following case analysis shows that, for every trace subset S and every agency A, there exists a strategy of A to characterize S in figure 4(a) iff there exists a strategy for A to characterize S in figure 4(b).
  - **Case 1a:** *ϕ is $\exists\emptyset.\phi_1$.* In this case, there is no strategy and the sets of traces imposed by no strategy in the two state graphs are both $\{\Box p, \Box q\}$. Thus, $\exists\emptyset.\phi_1$ cannot distinguish the trace sets of the two state graphs.
  - **Case 1b:** *ϕ is $\exists\{1\}.\phi_1$.* From figure 4, no matter what strategies agency $\{1\}$ may choose, the trace sets for the two game graphs are both $\{\Box p, \Box q\}$. Thus, $\exists\{1\}.\phi_1$ cannot distinguish the trace sets of the two state graphs.
  - **Case 1c:** *ϕ is $\exists\{2\}.\phi_1$.* This case is similar to case 1b.
  - **Case 1d:** *ϕ is $\exists\{3\}.\phi_1$.* This case is similar to case 1b.
  - **Case 1e:** *ϕ is $\exists\{1,2\}.\phi_1$.* Agency $\{1,2\}$ can cooperate to force three trace sets: $\{\Box p\}, \{\Box q\}, \{\Box p, \Box q\}$ in both of the game graphs in figure 4. Thus, $\exists\{1,2\}.\phi_1$ cannot distinguish the trace sets of the two state graphs.
  - **Case 1f:** *ϕ is $\exists\{1,3\}.\phi_1$.* This case is similar to case 1e.
  - **Case 1g:** *ϕ is $\exists\{2,3\}.\phi_1$.* This case is similar to case 1e.
  - **Case 1h:** *ϕ is $\exists\{1,2,3\}.\phi_1$.* Agency $\{1,2,3\}$ can cooperate to force three trace sets: $\{\Box p\}, \{\Box q\}, \{\Box p, \Box q\}$ in both of the game graphs in figure 4. Thus, $\exists\{1,2,3\}.\phi_1$ cannot distinguish the trace sets of the two state graphs.

- **Case 2:** *ϕ is a Boolean combination of formulas in case 1.* Since case 1 does not distinguish the two game graphs, this case can neither do it.

Thus the base case is proven.

Induction step: We use structural induction on GL formulas to prove the step. If ϕ has k modal operators, then we can use the two graphs in figure 5. It is clear that $\langle 1\rangle((\langle +2\rangle\Box p) \wedge (\langle +2\rangle\Box q))$ can distinguish the two graphs. However, to tell the difference between the two graphs, we need a modal subformula of ϕ that can tell the difference of G_{k-1} and H_{k-1}. But according to the inductive hypothesis, this is impossible. Thus the lemma is proven. ■

Lemma 6 *GL formula $\exists\!\!\exists\{1\}.((\exists\Box p) \wedge \exists\Box q)$ is not equivalent to any BSIL formula.*

Proof : The proof basically follows the same argument in [2] that $\exists\!\!\exists\{1\}.((\exists\Box p) \wedge \exists\Box q)$ is not equivalent to any ATL* formula. ■

Lemmas 5 and 6 together show that GL and BSIL are not comparable in expressiveness.

4.3 Comparision with AMC

AMC is an extension from μ-calculus and allows for multiple fixpoints interleaved together [2]. An AMC formula contains fixpoint operators on state set variables. The only modality of AMC is of the form $\langle A\rangle \bigcirc \psi$ and least fixpoint **lfp**$x.\psi_1(x)$ where $\psi_1(x)$ is a Boolean function of atomic propositions and state set variables (including x). It is required that every occurrence of x in ψ_1 is under even number of negations. The duality of the least fixpoint operator is the greatest fixpoint operator **gfp**. Formula **gfp**$x.\psi_1$ is defined as $\neg$**lfp**$x.\neg\psi_1(x)$.

To establish that AMC is not as expressive as BSIL, we basically follow the proof style for Lemma 3 and use the same two families of game graphs. The statement of the lemma requires notations for state set variables and other details in AMC. We need to define the domain of values for the free state set variables in AMC formulas. Let X be the set of state set variables. Without loss of generality, we assume that no two subformulas of the form **lfp**$x.\phi$ in a given AMC formula share the same quantified name of x. Given a subformula **lfp**$x_i.\phi$ with free variables $x_1, \ldots, x_n$ in ϕ and no modal operator $\langle \ldots\rangle\bigcirc$ in ϕ, we define ϕ as a *base template* for x_i. Then we can define the *base formula domain* of x_i, denoted $F_0(x_i)$, as the smallest set with the following restrictions. We let $\phi[x_1 \mapsto \eta_1, \ldots, x_n \mapsto \eta_n]$ be the AMC formula identical to ϕ except that every occurrence of x_i in ϕ are respectively replaced with η_i.

- For each $x_i \in X$ with base template ϕ, $\phi[x_1 \mapsto \mathit{false}, \ldots, x_n \mapsto \mathit{false}] \in F_0(x_i)$.
- For each $x_i \in X$ with base template ϕ and $\phi_1 \in F_0(x_1), \ldots, \phi_n \in F_0(x_n)$, $\phi[x_1 \mapsto \phi_1, \ldots, x_n \mapsto \phi_n] \in F_0(x_i)$.

Note that there is neither variables nor '**lfp**' operators in $F_0(x)$ for every x. Thus we can define the characterization κ of a formula ϕ_1 in $F_0(x)$ for $\mathcal{G}$, in symbols $\kappa(\mathcal{G}, \phi_1)$, as follows.

- For each atomic proposition $p \in P$, $\kappa(\mathcal{G}, p) = \{q \mid p \in \lambda(q)\}$.
- For each $\phi \in F_0(x)$, $\kappa(\mathcal{G}, \neg\phi) = Q - \kappa(\mathcal{G}, \phi)$.

- For each $\phi_1, \phi_2 \in F_0(x)$, $\kappa(\mathcal{G}, \phi_1 \vee \phi_2) = \kappa(\mathcal{G}, \phi_1) \cup \kappa(\mathcal{G}, \phi_2)$.

A *valuation* ν of variables in X for $\mathcal{G}$ is a mapping from X such that, for each $x \in X$, there exists a base domain formula ϕ of x such that $\nu(x) = \kappa(\mathcal{G}, \phi)$.

The expressiveness comparison between AMC and BSIL relies on the following lemma.

Lemma 7 *For every AMC formula ϕ without modal operator of the form $\langle \ldots \rangle \bigcirc$, state set variable x, and a formula ϕ_1 in $F_0(x)$, $r_0 \in \kappa(G_0, \phi_1)$ iff $s_0 \in \kappa(H_0, \phi_1)$ in figure 4.*

Proof : We can prove this with an structural induction on ϕ_1. The base case is straightforward. The inductive step follows since Boolean combinations of subformulas that cannot distinguish G_0 and H_0 can neither distinguish the two game graphs. ■

Now we want to classify AMC formulas according to the nesting depths of operators of the form $\langle \ldots \rangle \bigcirc$ in a formula. Specifically, we let AMC$^{(k)}$ be the set of AMC formulas with exactly nesting depth k of operator $\langle \ldots \rangle \bigcirc$. For example, $\mathbf{lfp}x.\langle 1 \rangle \bigcirc (p \rightarrow \mathbf{lfp}y.\langle 2 \rangle (x \vee y \wedge \bigcirc q))$ is in AMC$^{(2)}$. Then, AMC$^{(0)}$ is the smallest set with the following restrictions.

- **Case 1a:** For each atomic proposition $p \in P$, $p \in$ AMC$^{(0)}$.
- **Case 1b:** For each proposition variable $x \in X$, $x \in$ AMC$^{(0)}$.
- **Case 1c:** For each $\phi \in$ AMC$^{(0)}$, $\neg\phi \in$ AMC$^{(0)}$.
- **Case 1d:** For each $\phi_1, \phi_2 \in$ AMC$^{(0)}$, $\phi_1 \vee \phi_2 \in$ AMC$^{(0)}$.
- **Case 1e:** For each $\phi \in$ AMC$^{(0)}$ and proposition variable $x \in X$, $\mathbf{lfp}x.\phi \in$ AMC$^{(0)}$.

Then AMC$^{(k)}$, $k > 0$, is the smallest set with the following restrictions.

- **Case 2a:** For each $A \subseteq [1, m]$ and $\phi \in$ AMC$^{(k-1)}$, $\langle A \rangle \bigcirc \phi \in$ AMC$^{(k)}$,
- **Case 2b:** For each $\phi \in$ AMC$^{(k)}$, $\neg\phi \in$ AMC$^{(k)}$.
- **Case 2c:** For each $\phi_1 \in$ AMC$^{(k)}$ and $\phi_2 \in \bigcup_{h \leq k}$ AMC$^{(h)}$, $\phi_1 \vee \phi_2 \in$ AMC$^{(k)}$.
- **Case 2d:** For each $\phi_1 \in \bigcup_{h \leq k}$ AMC$^{(h)}$ and $\phi_2 \in$ AMC$^{(k)}$, $\phi_1 \vee \phi_2 \in$ AMC$^{(k)}$.
- **Case 2e:** For each $\phi \in$ AMC$^{(k)}$ and proposition variable $x \in X$, $\mathbf{lfp}x.\phi \in$ AMC$^{(k)}$.

Note that there could be free variables in the formulas classified in the above. The evaluation of such formulas for a game graph depends on the valuation of the free variables.

Given two game graphs G, H and an AMC formulas ϕ, we say that two valuations of ν and ν' respectively of G and H are *consistent* if for every $x \in X$, there exists a $\phi_1 \in F_0(x)$ such that $\nu(x) = \kappa(G, \phi_1)$ and $\nu'(x) = \kappa(H, \phi_1)$. In the following, we adopt the AMC semantic notations in [2]. Given a game graph G, an AMC formula ϕ, and a valuation ν of state set variables in X, $(\phi)^G(\nu)$ denotes the set of states of G that satisfy ϕ with valuation ν.

Lemma 8 *Assume that G_k and H_k are defined in figures 4 and 5. For every k, AMC formula $\phi \in AMC^{(k)}$, and two consistent valuations ν and ν' respectively of G_k and H_k, $r_k \in (\phi)^{G_k}(\nu)$ iff $s_k \in (\phi)^{H_k}(\nu')$.*

Proof : We use an induction on k to prove the lemma.

Base case: When ϕ is in case 1a through 1d, the lemma follows straightforwardly. In case 1e, $(\mathbf{lfp}x.\psi)^{G_0}(\nu)$ can be expanded as follows. We let

- $\psi^{G_0,\nu,(0)}$ be $(\psi[x \mapsto \mathit{false}])^{G_0}(\nu)$ and
- for each $h > 0$, $\psi^{G_0,\nu,(h)}$ be $(\phi[x \mapsto \psi_1^{G_0,(h-1)}])(\nu)$.

Then $(\mathbf{lfp}x.\phi)^{G_0}(\nu) = \bigcup_{h\geq 0} \psi^{G_0,\nu,(h)}$ according to the semantics of AMC. Similarly, $(\mathbf{lfp}x.\phi)^{H_0}(\nu') = \bigcup_{h\geq 0} \psi^{H_0,\nu',(h)}$. According to the same argument for cases 1a through 1d, for each $h \geq 0$, $r_0 \in \psi^{G_0,\nu,(h)}$ iff $s_0 \in \psi^{H_0,\nu',(h)}$. Thus it is clear that $r_0 \in (\mathbf{lfp}x.\phi)^{G_0}(\nu)$ iff $s_0 \in (\mathbf{lfp}x.\phi)^{H_0}(\nu')$. Thus the lemma is proven in this case.

Induction step: To tell the difference between G_k and H_k, we need a formula with the following structure.

- At least one nesting of operators like $\langle \ldots \rangle\bigcirc$ in a least fixpoint operation to infer the reachability of G_{k-1} and H_{k-1}.
- A modal subformula nested inside a $\langle \ldots \rangle\bigcirc$ modal operator of ϕ that can tell the difference of G_{k-1} and H_{k-1}. But according to the inductive hypothesis, this is impossible.

Thus the lemma is proven. ■

Then with Lemma 8, we conclude the proof for Lemma 9 in the following.

Lemma 9 *For every AMC formula ϕ, there are two game graphs that ϕ cannot distinguish while $\langle 1\rangle((\langle +2\rangle\Box p) \wedge \langle +2\rangle\Box q)$ can.*

Proof : In the proof for Lemma 8, it is apparent that ϕ with $\phi \in \text{AMC}^{(k)}$ cannot tell G_k and H_k. ■

By the same argument in [2], for one-agent game, BSIL coincides with CTL and is not as expressive as AMC.

Lemma 10 *For game graphs of one agent, AMC is strictly more expressive than BSIL.*

Proof : For one-agent games, AMC is equivalent to μ-calculus and BSIL is equivalent to CTL which is strictly less expressive than μ-calculus. ■

A comment on Lemmas 3, 5, and 9 is that the path modal formulas in the lemmas can be changed independently to $\Diamond\neg p$ and $\Diamond\neg q$ without affecting the validity of the lemma. This can be used to show that the example properties in the introduction are indeed inexpressible in ATL*, GL, and AMC.

5 BSIL and ATL$^+$ model-checking are PSPACE complete

The model-checking problem of BSIL is contained in PSPACE mainly due to the restriction that disallows negation in tree formulas. As in the model-checking algorithms of ATL [2], we can evaluate the proper state subformulas formulas independently and then treat them as auxiliary propositions. Moreover, as in the evaluation of $\Diamond$-formulas in ATL model-checking, if a $\Diamond$-formula can be enforced with an S-profile, it can be enforced in a finite number of steps along every play compatible with the strategy in a computation tree. Once a bound b for this finite number of

steps is determined, we can enumerate all strategies embedded in the computation tree up to depth b and try to find one that enforces a BSIL formula.

There are, however, severe differences between exploring a computation tree for BSIL and exploring one for ATL [2]. For BSIL, we have to take the interaction of strategies into account. For example, we may have to enforce a subformula $\langle 1\rangle((\langle +2\rangle \Diamond p) \wedge \langle +2\rangle(\Box q \vee \Diamond r))$. Then, when exploring the computation tree, we may follow two strategies of Agent 2, one to enforce $\Diamond p$ and the other to enforce $\Box q$ or $\Diamond r$. There are the following situation for the interaction between these two strategies. The two strategies may make the same decision all the way until we reach a tree node v. (For turn-based games, v has to be owned by Agent 2.) This can be conceptualized as passing the obligations of $\Diamond p$ and $\Box q \vee \Diamond r$ along the path from the root to v. Then, at node v, the two strategies may differ in their decisions and pass down the two obligations to different branches. In Subsection 5.1, we explain some basic concepts in labeling the children with path obligations passed down from their parent in a computation tree to obey the interaction among the strategies declared in a BSIL formula.

Then, in Subsection 5.2, we present our algorithm in two parts, one for model-checking BSIL state formulas and the other for model-checking BSIL tree formulas. In Subsection 5.3, we prove the correctness of the algorithm. In Subsection 5.4, we show that our algorithm is in PSPACE. Together with Lemma 17 in Subsection 5.5, we then establish the PSPACE-completeness of the BSIL and ATL$^+$ model-checking problems.

5.1 Computing path obligations and passing them down the computation tree

We use $\{a_1 \mapsto s_1, \ldots, a_n \mapsto s_n\}$ to denote a partial function that maps a_i to s_i for each $i \in [1, n]$. Given a partial function f, we denote the domain of f by *def*(f).

We need some special techniques in checking tree formulas. We adopt the concept of strategy variables from [5, 15]. A *strategy variable binding* (*SV-binding* for short) is a partial function from $[1, m]$ to strategy variables. Given an SV-binding Λ, $\Lambda \circ \{a_1 \mapsto s_1, \ldots, a_n \mapsto s_n\}$ is the SV-binding that is identical to Λ except that Agent a_i is bound to s_i for every $i \in [1, n]$.

Suppose that we are given SV-bindings $\Lambda_1, \ldots, \Lambda_n$ and S-profiles $\Sigma_1, \ldots, \Sigma_n$. We say that $\Lambda_1, \ldots, \Lambda_n$ *matches* $\Sigma_1, \ldots, \Sigma_n$ if, and only if, for every $a \in [1, m]$ and $i, j \in [1, n]$ with $a \in$ *def*$(\Sigma_i) \cap$ *def*(Σ_j), $\Sigma_i(a) = \Sigma_j(a)$ if, and only if, $\Lambda_i(a) = \Lambda_j(a)$.

Given an SV-binding Λ and a state, tree, or path formula ψ, $\Lambda\psi$ is called a *bound formula*. $\Lambda\psi$ is a *bound path obligation* (*BP-constraint*) if ψ is a Boolean combination of path formulas. A Boolean combination of BP-obligations is called a *Boolean bound formula* (*BB-formula*). The strategy variables in BB-formula are only used to tell whether or not two path properties are to be enforced with the same strategy. For example, the property $\langle 1\rangle((\langle +2\rangle \Diamond p) \wedge \langle +2\rangle(\Box q \vee \Diamond r))$ can be rewritten as BB-formula $(\{1 \mapsto s_1, 2 \mapsto s_2\}\Diamond p) \wedge \{1 \mapsto s_1, 2 \mapsto s_3\}\Box q \vee \Diamond r$, which says that Agent 1 must use the same strategy to fulfill both $\Diamond p$ and $\Box q \vee \Diamond r$, while Agent 2 may use different strategies to fulfill these two path properties.

Table 1: Rewriting rules for BB-formulas

$bf(\Lambda\neg\neg\phi)$	$\equiv$	$bf(\Lambda\phi)$
$bf(\Lambda(\tau_1 \vee \tau_2))$	$\equiv$	$bf(\Lambda\tau_1) \vee bf(\Lambda\tau_2)$
$bf(\Lambda(\tau_1 \wedge \tau_2))$	$\equiv$	$bf(\Lambda\tau_1) \wedge bf(\Lambda\tau_2)$
$bf(\Lambda\langle a_1, \ldots, a_n\rangle\psi)$	$\equiv$	$bf(\{a_1 \mapsto newVar(), \ldots, a_n \mapsto newVar()\}\psi)$
$bf(\Lambda\langle +a_1, \ldots, a_n\rangle\psi)$	$\equiv$	$bf(\Lambda \circ \{a_1 \mapsto newVar(), \ldots, a_n \mapsto newVar()\}\psi)$
$bf(\Lambda \bigcirc \phi_1)$	$\equiv$	$\Lambda \bigcirc bf(\emptyset\phi_1)$
$bf(\Lambda\neg \bigcirc \phi_1)$	$\equiv$	$\Lambda \bigcirc bf(\emptyset\neg\phi_1)$
$bf(\Lambda\phi_1 \mathrm{U} \phi_2)$	$\equiv$	$\Lambda bf(\emptyset\phi_1)\mathrm{U} bf(\emptyset\phi_2)$
$bf(\Lambda\neg\phi_1 \mathrm{U} \phi_2)$	$\equiv$	$\Lambda((bf(\emptyset\phi_1)\mathrm{U} bf(\neg\emptyset(\phi_1 \vee \phi_2))) \vee \Box bf(\emptyset\neg\phi_2))$
$bf(\Lambda p) \equiv p$	;	$bf(\Lambda\neg p) \equiv \neg p$
$bf(\Lambda true) \equiv true$	;	$bf(\Lambda\neg true) \equiv false$
$bf(\Lambda false) \equiv false$	;	$bf(\Lambda\neg false) \equiv true$

ϕ_1, ϕ_2: state or path formulas. τ_1, τ_2: tree formulas. ψ_1, ψ_2: tree or path formulas.

Suppose we are given a function π that maps symbolic strategy names to strategies. Similar to the semantics of strategy logics [15] with strategy variables, we can also define the satisfaction of BB-formulas $\Lambda\psi$ at a state q with π, in symbols $\mathcal{G}, q \models^{\pi} \Lambda\psi$, as follows.

- $\mathcal{G}, q \models^{\pi} \Lambda_1\psi_1 \vee \Lambda_2\psi_2$ iff $\mathcal{G}, q \models^{\pi} \Lambda_1\phi_1$ or $\mathcal{G}, q \models^{\pi} \Lambda_2\phi_2$ holds.
- $\mathcal{G}, q \models^{\pi} \Lambda_1\phi_1 \wedge \Lambda_2\phi_2$ iff both $\mathcal{G}, q \models^{\pi} \Lambda_1\phi_1$ and $\mathcal{G}, q \models^{\pi} \Lambda_2\phi_2$ hold.
- Given an SV-binding Λ and a path formula ψ_1 with an S-profile $\Sigma = \{a \mapsto \pi(\Lambda(a)) \mid a \in def(\Lambda)\}$, $\mathcal{G}, q \models^{\pi} \Lambda\psi_1$ iff, for all plays ρ compatible with Σ from q, $\rho \models_{\Sigma} \psi_1$ holds.

In Table 1, we present equivalence rules to rewrite state, tree, and path formulas to BB-formulas using the procedure *bf*(). For convenience, we use a procedure *newVar*() that returns a strategy variable that has not been used before. In general, the semantics of BSIL deals with the satisfaction of a set of subformulas bound to different S-profiles. The following two lemmas relate the rules in Table 1 with the semantics of BSIL formulas.

Lemma 11 *Suppose we are given a state q, BSIL subformulas $\psi_1, \ldots, \psi_n$, and S-profiles $\Sigma_1, \ldots, \Sigma_n$ such that, for all $i \in [1, n]$, $\mathcal{G}, q \models_{\Sigma_i} \psi_i$. Then there exist SV-bindings $\Lambda_1, \ldots, \Lambda_n$ that match $\Sigma_1, \ldots, \Sigma_n$ and a function π such that for all $i \in [1, n]$, $\mathcal{G}, q \models^{\pi} bf(\Lambda_i\psi_i)$.*

Proof : First, it is obvious that we can define $\Lambda_1, \ldots, \Lambda_n$ that match $\Sigma_1, \ldots, \Sigma_n$. Then the lemma can be proven by defining, for each $a \in [1, m]$ and $i \in [1, n]$ with $a \in def(\Lambda_i)$, that $\pi(\Lambda_i(a)) = \Sigma_i(a)$. ■

Lemma 12 *Suppose we are given a state q, BSIL subformulas $\psi_1, \ldots, \psi_n$, SV-bindings $\Lambda_1, \ldots, \Lambda_n$, and a function π such that, for all $i \in [1, n]$, $\mathcal{G}, q \models^{\pi} bf(\Lambda_i\psi_i)$. Then there exist S-profiles $\Sigma_1, \ldots, \Sigma_n$ such that, for all $i \in [1, n]$, $\mathcal{G}, q \models_{\Sigma_i} \psi_i$.*

Proof : For each $i \in [1,n]$, we can construct Σ_i by defining, for all $a \in \mathit{def}(\Lambda_i)$, $\Sigma_i(a) = \pi(\Lambda_i(a))$. Then by a structural induction on $\psi_1, \ldots, \psi_n$, we can prove the lemma. ■

To ease the presentation of our algorithms, we also assume that there is a procedure that rewrites a BB-formula to an equivalent BB-formula in disjunctive normal form. Specifically, a *disjunctive normal BB-formula* (*DNBB-formula*) is the disjunction of conjunctions of BP-obligations. The rewriting of a BB-formula ϕ to a DNBB-formula can be done by repeatedly applying the distribution law of conjunctions of disjunctions until a DNBB-formula is obtained.

Example 2 *DNBB-formula rewriting:* We have the following rewriting process for a BSIL formula for five agents.

$$
\begin{array}{ll}
& \mathit{bf}(\emptyset\langle 1,2\rangle\,(\langle +3\rangle(\Box p \vee \Diamond q) \wedge \langle +3\rangle(\langle +2\rangle\Diamond r \vee \langle +4\rangle\Box q))) \\
\equiv & \mathit{bf}(\{1\mapsto s_1, 2\mapsto s_2\}\,(\langle +3\rangle(\Box p \vee \Diamond q) \wedge \langle +3\rangle(\langle +2\rangle\Diamond r \vee \langle +4\rangle\Box q))) \\
\equiv & \mathit{bf}(\{1\mapsto s_1, 2\mapsto s_2\}\langle +3\rangle(\Box p \vee \Diamond q)) \wedge \mathit{bf}(\{1\mapsto s_1, 2\mapsto s_2\}\langle +3\rangle(\langle +2\rangle\Diamond r \vee \langle +4\rangle\Box q)) \\
\equiv & \mathit{bf}(\{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_3\}(\Box p \vee \Diamond q)) \wedge \mathit{bf}(\{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_4\}(\langle +2\rangle\Diamond r \vee \langle +4\rangle\Box q)) \\
\equiv & \{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_3\}(\Box p \vee \Diamond q) \\
& \wedge (\{1\mapsto s_1, 2\mapsto s_5, 3\mapsto s_4\}\Diamond r \vee \{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_4, 4\mapsto s_6\}\Box q) \\
\equiv & (\{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_3\}(\Box p \vee \Diamond q) \wedge \{1\mapsto s_1, 2\mapsto s_5, 3\mapsto s_4\}\Diamond r) \\
& \vee (\{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_3\}(\Box p \vee \Diamond q) \wedge \{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_4, 4\mapsto s_6\}\Box q)
\end{array}
$$

This DNBB-formula sheds some light on the analysis of BSIL formulas. As can be seen, the formula is satisfied iff one of the two outermost disjuncts is satisfied. Without loss of generality, we examine the first disjunct:

$$\eta_1 \equiv \{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_3\}(\Box p \vee \Diamond q) \wedge \{1\mapsto s_1, 2\mapsto s_5, 3\mapsto s_4\}\Diamond r$$

There are the following two S-profiles involved in the satisfaction of the formula.

- Σ_1 for $\{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_3\}$ of $\{1,2,3\}$ used to satisfy $\Box p \vee \Diamond q$.
- Σ_2 for $\{1\mapsto s_1, 2\mapsto s_5, 3\mapsto s_4\}$ of $\{1,2,3\}$ used to satisfy $\Diamond r$.

This disjunct imposes the restrictions that Σ_1 and Σ_2 must agree in their moves by agent 1. (Or for turn-based games, they must agree in their choices at nodes owned by Agent 1.) Similarly, we can examine

$$\eta_2 \equiv \{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_3\}(\Box p \vee \Diamond q) \wedge \{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_4, 4\mapsto s_6\}\Box q$$

There is a new S-profile introduced.

- Σ_3 for $\{1\mapsto s_1, 2\mapsto s_2, 3\mapsto s_4, 4\mapsto s_6\}$ of $\{1,2,3,4\}$ used to satisfy $\Box q$.

This disjunct imposes the restrictions that Σ_1 and Σ_3 must agree in their moves by Agents 1 and 2. In the following, we use the observation in this example to construct structures from DNBB-formulas for the model-checking of conjunctive DNBB-formulas. ■

For the ease of notation, we represent a conjunctive DNBB-formula η as a set of BP-obligations in our algorithms. Our goal is to design a computation tree exploration procedure that given a set C of BP-obligations, labels each node in the tree with a subset of C for the set of path formulas that some S-profiles have to enforce without

violating the restrictions of strategy interaction imposed in C through the strategy variables. In the design of the procedure, one central component is how to label the children of a node with appropriate sets of BP-obligations as inherited path obligations from C. We need two basic procedures for this purpose. The first is to evaluate the truth values of path literals in BP-obligations with a proposition interpretation when possible. Specifically, when we can deduce the truth values of U-formulas and $\Box$-formulas from the truth values of propositions (or state subformulas) at a state, the procedure changes the respective U-formula and $\Box$-formula to their respective truth values. The procedure is as follows.

$\texttt{eval}(W, \theta)$ // $\lambda()$ has been extended with satisfied state subformulas at each state.

1: **switch** (θ)
2: **case** *true* or *false*: **return** θ
3: **case** p: **if** $p \in W$ **then return** *true* **else return** *false* **end if**
4: **case** $\neg p$: **if** $p \in W$ **then return** *false* **else return** *true* **end if**
5: **case** $\theta_1 \vee \theta_2$:
6: Let θ_1 be $\texttt{eval}(W, \theta_1)$ and θ_2 be $\texttt{eval}(W, \theta_2)$.
7: **if** θ_1 is *true* or θ_2 is *true* **then return** *true*.
8: **else if** θ_1 is *false* **then return** θ_2. **else if** θ_2 is *false* **then return** θ_1. **else return** $\theta_1 \vee \theta_2$.
9: **end if**
10: **case** $\theta_1 \wedge \theta_2$:
11: Let θ_1 be $\texttt{eval}(W, \theta_1)$ and θ_2 be $\texttt{eval}(W, \theta_2)$.
12: **if** θ_1 is *false* or θ_2 is *false* **then return** *false*.
13: **else if** θ_1 is *true* **then return** θ_2. **else if** θ_2 is *true* **then return** θ_1. **else return** $\theta_1 \wedge \theta_2$.
14: **end if**
15: **case** $\bigcirc\phi_1$: **return** θ
16: **case** $\Box\phi_1$: **if** $\phi_1 \notin W$ **then return** *false* **else return** θ **end if**
17: **case** $\phi_1 \mathrm{U} \phi_2$: **if** $\phi_2 \in W$ **then return** *true* **else if** $\phi_1 \notin P$ **then return** *false* **else return** θ **end if**
18: **end switch**

Statements 17 checks if $\Lambda\theta$ is fulfilled. When it is fulfilled, θ is changed to *true*. Statements 17 and 16 also check if $\Lambda\theta$ is violated. When a violation happens, θ is changed to *false*.

Then we need a procedure, $\texttt{next}()$, that calculates the BP-obligations passed down from a previous state. This is simply done by replacing every $\bigcirc\phi$ by ϕ in the BP-obligations.

With the two basic procedures defined above, we now present a procedure that nondeterministically calculates sets of BP-obligations passed down to the successor states. This is accomplished with the procedure *sucSet*(q, C) in the following. Given a node q in the computation tree and a set C, the procedure nondeterministically returns an assignment of BP-obligations to children of q to enforce the BP-obligations in C without violating the strategy

interaction of BP-obligations.

---

sucSet(q, C) // $\lambda()$ has been extended with satisfied state subformulas at each state.

1: Convert C to $\{\Lambda\texttt{eval}(\lambda(q), \theta) \mid \Lambda\theta \in C\}$.
2: **if** Λ*false* $\in C$ **then return** $\emptyset$ **end if**
3: Let S be the set of all symbolic strategy variables in C. That is, $S = \{s \mid a \mapsto s \in \Lambda, \Lambda\theta \in C\}$.
4: Nondeterministically pick an $\alpha_s \in \Delta$ for each $s \in S$.
5: Let K be $\{(q', \emptyset) \mid (q, q') \in R\}$.
6: **for** each $\Lambda\theta \in C$ **do**
7: **if** for all (q, q'), there is an $a \in$ *def*(Λ) with $\delta((q, q'), a) \neq s_{\Lambda(a)}$ **then** return $\emptyset$ **end if**
8: **for** $(q', C') \in \Delta$ with $\forall a \in$ *def*$(\Lambda)(\delta((q, q'), a) = s_{\Lambda(a)})$ **do**
9: Replace (q', C') with $(q', C' \cup \{\Lambda\texttt{next}(\theta)\})$ in K.
10: **end for**
11: **end for**
12: **return** K.

The nondeterministic choices at statement 4 make sure that one symbolic strategy variable is mapped to exactly one move. The loop at statement 6 iterates through all the path obligations at the current node and passes them down to the children if necessary. The if-statement at line 7 checks whether all obligations can be passed down to some children. If some obligations are not passed due to mismatch between moves of the strategies and the labels on the transitions, then the we return with failure. Otherwise, statement 8 passes the obligations to all children with matching transition labels. The obligations to children are recorded in K which is returned with success at statement 12.

5.2 Procedures for checking BSIL properties

The procedure in the following checks a BSIL state property ϕ at a state q of A.

---

checkBSIL(q, ϕ)

1: **if** ϕ is p **then if** $\phi \in \lambda(q)$ **then return** *true*. **else return** *false*. **end if**
2: **else if** ϕ is $\phi_1 \vee \phi_2$ **then return** checkBSIL$(q, \phi_1) \vee$ checkBSIL(q, ϕ_2)
3: **else if** ϕ is $\neg\phi_1$ **then return** $\neg$checkBSIL(q, ϕ_1)
4: **else if** ϕ is $\langle A\rangle\tau$ for a tree or path formula τ **then return** checkTree$(q, \langle A\rangle\tau)$
5: **end if**

The procedure is straightforward and works inductively on the structure of the input formula. For convenience, we need procedure checkSetOfBSIL$(Q'\phi_1)$ in the following that checks a BSIL property ϕ_1 at each state in Q'.

---

checkSetOfBSIL(Q', ϕ_1)

1: **if** $\phi_1 \notin P \cup \{\mathit{true}, \mathit{false}\}$ **then**
2: **for** each $q' \in Q'$ **do**
3: **if** `checkBSIL`(q', ϕ_1) **then** Let $\lambda(q')$ be $(\lambda(q') \cup \{\phi_1\}) - \{\neg\phi_1\}$.
4: **else** Let $\lambda(q')$ be $(\lambda(q') - \{\phi_1\}) \cup \{\neg\phi_1\}$. **end if**
5: **end for**
6: **end if**

Then we use procedure `checkTree`$(q, \langle A\rangle\tau)$ in the following to check if a state q satisfies $\langle A\rangle\tau$.

`checkTree`$(q, \langle A\rangle\tau)$

1: Rewrite *bf*$(\emptyset\langle A\rangle\tau)$ to DNBB-formula $\eta_1 \vee \ldots \vee \eta_n$.
2: **for** $i \in [1, n]$ **do**
3: Represent η_i as a set C of BP-obligations.
4: **for** each $\Lambda\theta$ in C. **do**
5: **if** θ is $\bigcirc\phi_1$ **then** `checkSetOfBSIL`$(\{q' \mid (q, q') \in \mathcal{R}\}, \phi_1)$.
6: **else if** θ is $\phi_1 \mathrm{U} \phi_2$ **then** `checkSetOfBSIL`(Q, ϕ_1); `checkSetOfBSIL`(Q, ϕ_2); **end if**
7: **end for**
8: **if** `recTree`(q, C) **then return** *true*. **end if**
9: **end for**
10: **return** *false*.

We first rewrite $\langle A\rangle\tau$ to its DNBB-formula at statement 1 by calling *bf*$(\emptyset\langle A\rangle\tau)$ and using the distribution law of conjunctions over disjunctions. (In practice, to contain the complexity in PSPACE, we only need to enumerate the disjuncts of the DNBB-formula in PSPACE.) We then iteratively check with the loop starting from statement 2 if $\langle A\rangle\tau$ is satisfied due to one of its conjunctive DNBB-formula components of $\langle A\rangle\tau$. At statement 3, we construct the set C of BP-obligations of the component. We evaluate the subformulas with the inner loop starting at statement 4. Finally at statement 8, we explore the computation tree, with procedure `recTree`(q, C) in the following, and pass down the path obligations to the children according to the restrictions of the SV-binding in C.

`recTree`(q, C)

1: **if** (q, C) coincides with an ancestor in the exploration **then**
2: **if** there is no $\Lambda\phi_1 \mathrm{U} \phi_2$ in C **then return** *true*; **else return** *false*. **end if**
3: **end if**
4: **if** *sucSet*(q, C) is empty **then return** *false* **end if**
5: **for** each $(q', C') \in$ *sucSet*(q, C) with $C' \neq \emptyset$ **do**
6: **if** `recTree`(q', C') is *false* **then return** *false*. **end if**
7: **end for**

8: **return** *true*.

Note that procedure $\texttt{recTree}(q, C)$ is nondeterministic since it employs *sucSet*(q, C) to nondeterministically calculate an assignment Δ of path obligations to the children of q.

5.3 Correctness proof of the algorithm

In order to prove the correctness of this algorithm, we define *obligation distribution trees* (*OD-trees*) in the following. An OD-tree for a set C of BP-obligations and game graph $\mathcal{G}$ from a state $q_0 \in Q$ is a labeled computation tree $\langle V, \bar{r}, \alpha, E, \beta \rangle$ with the following restrictions.

- V is the set of nodes in the tree.
- $\bar{r} \in V$ is the root of the tree.
- $\alpha : V \mapsto Q$ labels each tree node with a state. Also $\alpha(\bar{r}) = q_0$.
- $E \subseteq V \times V$ is the set of arcs of the tree such that, for each $(q, q') \in R$, there exists an $(v, v') \in E$ with $\alpha(v) = q$ and $\alpha(v') = q'$.
- $\beta : V \mapsto 2^C$ labels each node with a subset of C for path formulas in χ that need to be fulfilled at a node. Moreover, we have the following restrictions on β.
  - $C \subseteq \beta(r)$.
  - For every $v \in V$, there exists a $\Delta =$ *sucSet*$(\alpha(v), \beta(v))$ such that, for every $(q', C') \in \Delta$, there exists a $(v, v') \in E$ with $\alpha(v') = q'$ and $\beta(v') = C'$.

The OD-tree is *fulfilled* iff, for every path $v_0 v_1 \dots v_k \dots$ along the tree from the root, there exists an $h \geq 0$ such that, for every $j \geq h$, there is no $\Lambda\phi_1 \mathrm{U} \phi_2 \in \beta(v_j)$. We have the following connection between an OD-tree and an execution of procedure $\texttt{recTree}(q, C)$ from the root of an OD-tree.

Lemma 13 *For a set C of BP-obligations, $\texttt{recTree}(q, C)$ returns true iff there exists a fulfilled OD-tree for C and $\mathcal{G}$ from q.*

Proof : In order to prove the lemma, we show both directions.

($\Rightarrow$) : It is straightforward to see that $\texttt{recTree}(q, C)$ returns true only if a finite tree has been constructed with leafs duplicating their ancestors. According to statement 2 of $\texttt{recTree}(q, C)$, it is clear that along the path from that ancestor to a leaf, no node is labeled with a BP-obligation of the form $\Lambda\phi_1 \mathrm{U} \phi_2$ by β. Thus we can extend the leaves by duplicating the subtree rooted at their duplicating ancestors. In this way, we can extend the finite tree to a fulfilled OD-tree.

($\Leftarrow$) : Suppose there exists a fulfilled OD-tree for C and $\mathcal{G}$ from q. Since all infinite paths from the root stabilize to suffices without index labels by β for until-formulas (as the tree is finitely branching, it would otherwise contain an infinite path with standing untility by Köngs lemma), we can repeatedly replace every subtree T with a subtree T' of T such that the root of T and T' have the same α and β labels. We can repeat this replacement until no node labeled

with an until-formula has the same α and β labels as one of its descendants. The existence of such an OD-tree after the replacements implies that $\texttt{recTree}(q, C)$ eventually explores such a tree, finds the termination condition at all leafs, and returns *true*. ■

Lemma 14 *Given a conjunctive DNBB-formula η represented as a set C of BP-obligations, there exists a function π on strategy variables in η with $\mathcal{G}, q \models^{\pi} \eta$ iff there exists a fulfilled OD-tree for $\mathcal{G}$ and C from q.*

Proof : The lemma can also be proven in two directions. In the forward direction, we can use π to construct S-profiles to enforce η. The S-profiles can then be used to construct a fulfilled OD-tree for $\mathcal{G}$ and C from q.

In the backward direction, we can follow the paths and obligations that are passed-down in the OD-tree and construct S-profiles that enforce η. Then, from these S-profiles, due to the one-to-one correspondence between the strategy variables and the strategies in the S-profiles, we can define a π with $\mathcal{G}, q \models^{\pi} \eta$. ■

The correctness of procedure $\texttt{recTree}(q, C)$ then directly follows from Lemmas 13 and 14. Then the correctness of procedure $\texttt{checkBSIL}(q, \phi)$ follows by a structural induction on a given BSIL formula and the correctness of procedure $\texttt{recTree}(q, C)$.

Lemma 15 *Given a state q in $\mathcal{G}$, $\texttt{checkBSIL}(q, \chi)$ iff $\mathcal{G}, q \models_{\perp} \chi$.* ■

5.4 Complexities of the algorithm

The algorithm that we presented in Subsections 5.1 and 5.2 can run in PSPACE mainly because we can enumerate the conjunts in a DNF in PSPACE and can implement procedure $\texttt{recTree}(q, C)$ with a stack of polynomial height. To see this, please recall that we use the procedure *sucSet*(q, C) to calculate the assignment of BP-obligations to the children to q in the computation tree. Specifically, procedure *sucSet*(q, C) nondeterministically returns a set Δ with elements of the form (q', C') such that $(q, q') \in R$ and $C' \subseteq C$. Thus, along any path in the OD-tree, the sets of literal bounds never increase. Moreover, when there is a node in the exploration of OD-tree that coincides with an ancestor, we backtrack in the exploration. This implies that, along any path segment longer than $|Q|$, one of the following two conditions hold.

- A backtracking happens at the end of the segment.
- The sets of BP-obligations along the segment must decrease in size at least once.

These conditions lead to the observation that, with procedure *sucSet*(q, C), the recursive exploration of a path can grow no longer than $1 + |C| \cdot |Q|$. This leads to the following lemma.

Lemma 16 *The BSIL model-checking algorithm in Subsections 5.1 and 5.2 is in PSPACE.*

Proof : For convenience, we let $\#(\chi)$ be the number of modal formulas in χ. Following the argument from above, it is straightforward to check that to explore an OD-tree, we only need a stack of at most $1 + \#(\chi) \cdot |Q|$ frames. In each frame, we only need to record a state in Q, a subset of $[1, \#(\chi)]$ for the path obligations, and a Δ returned

from procedure *sucSet*(q, C). Procedure *sucSet*(q, C) can be nondeterministically computed by randomly assigning the obligations in C to the successors of q and check if the assignment satisfies strategy interaction restriction of the BP-obligations. Procedure `checkBSIL`(q, ϕ) can then be executed in space cubic in the size of ϕ for the Δ's at nodes along the path. Thus, we conclude that the algorithm is a PSPACE algorithm. ■

A rough analysis of the time complexity of our algorithm follows. Let $|\chi|$ be the length of a BSIL formula χ. At each call to *sucSet*(), the size of C is at most $|\chi|$. The number of root-to-leaf paths in an OD-tree is at most $|\chi|$ since we only have to pass down $|\chi|$ BP-obligations. We can use the positions of the common ancestors of the leaves of such paths to analyze the number of the configurations of such OD-trees. The common ancestors can happen anywhere along the root-to-leaf paths. Thus, there are $(1 + |\chi| \cdot |Q|)^{|\chi|}$ ways to arrange the positions of the common ancestors since the length of paths are at most $1 + |\chi| \cdot |Q|$. The number of ways that the BP-obligations can be assigned to the leaves is at most $|\chi|^{|\chi|}$. The number of state labeling of the nodes on the paths is at most $|Q|^{|\chi|\cdot(1+|\chi|\cdot|Q|)}$. Thus, given a C, the total number of different OD-trees is in $O(|Q|^{|\chi|\cdot(1+|\chi|\cdot|Q|)}|\chi|^{|\chi|}(1 + |\chi| \cdot |Q|)^{|\chi|}) = O(|Q|^{|\chi|\cdot(2+|\chi|\cdot|Q|)}|\chi|^{2|\chi|})$. There are $O(2^{|\chi|})$ different possible values of C. There are at most $|\chi|$ OD-trees to construct for the model-checking task. Thus, the total time complexity of our algorithm is in $O(|\chi|2^{|\chi|}|Q|^{|\chi|\cdot(2+|\chi|\cdot|Q|)}|\chi|^{2|\chi|})$.

5.5 Lower bound and completeness

We close by establishing the PSPACE lower bound for ATL$^+$ model-checking, and hence the PSPACE-completeness of ATL$^+$ and BSIL model-checking. This is done by reduction from the prenex QBF (quantified Boolean formula) satisfiability problem [11] to an ATL$^+$ model-checking problem for a 2-agent game graph, where both the game graph and the ATL$^+$ specification are linear in the prenex QBF problem we reduce from. We assume a QBF property $\eta \equiv \bigtriangledown_1 p_1 \ldots \bigtriangledown_l p_l(C_1 \wedge C_2 \wedge \ldots \wedge C_n)$ with a set $P = \{p_1, \ldots, p_l\}$ of atomic propositions and the following restrictions.

- For each $k \in [1, l]$, $\bigtriangledown_k$ is either $\exists$ or $\forall$.
- For each $k \in [1, n]$, C_k is a clause $l_{k,1} \vee \ldots \vee l_{k,h_k}$, where, for each $j \in [1, h_k]$, $l_{k,j}$ is a *literal*, i.e., either an atomic proposition or a negated atomic proposition.

Intuitively, the reduction is to translate the QBF formula to a game graph and an ATL$^+$ formula for a traversal requirement on the game graph. The atomic propositions are then encoded as path constraints on the game graph. The interesting thing about the reduction is that the ATL$^+$ formula (naturally) contains no SIQs at all. This reduction may be interpreted as that we get the SIQ in the expressiveness without paying extra computation complexity.

Suppose that Γ_p represents the subgraphs for the truth of an atomic proposition p. The rest of the game graph is partitioned into subgraphs Ω_p responsible for the interpretation of atomic proposition p for all $p \in P$. Then the prenex QBF formula actually can be interpreted as a requirement for covering those Γ_p's with the decisions in those Ω_p's. For example, the following formula $\eta \equiv \exists p \forall q \exists r((p \vee q \vee r) \wedge (\neg p \vee \neg r))$ can be read as "*there exists a*

decision in Ω_p such that for every decision in Ω_q, there exists a decision in Ω_r such that

- *one of Γ_p, Γ_q, and Γ_r is covered; and*
- *either Γ_p or Γ_r is not covered.*

The details of constructing those Γ_p's and Ω_p's can be found in the proof of the following lemma that establishes the PSPACE complexity lower-bound.

Lemma 17 *The ATL$^+$ model-checking problem for turn-based game graphs is PSPACE-hard.*

Proof : Suppose we are given a prenex QBF η with propositions in P. We assume without loss of generality that all propositions are bound variables. (Note that, for the satisfiability problem, we can simply bind all free propositions by leading existential quantifiers.) We also use P for the atomic proposition set of $\mathcal{G}$. The idea is to use, for each $p \in P$, "$\Diamond p$" to encode that p is true, while "$\Box\neg p$" is used to encode that p is false. (Note that $\Box\neg p \equiv \neg\Diamond p$.) Then we construct a two-agent turn-based game graph G_η as shown in Figure 6, which reflects the structure of η: there is a sequence of (true) decisions (the states u_i and the state v_{n+1} have only one outgoing transition, and no true decision is taken there), which refer to the truth of the individual $\Box p_i$. These decisions are taken in the order given by the prenex quantifiers of η, and the existential decisions are taken by Agent 1 while the universal decisions are taken by Agent 2.

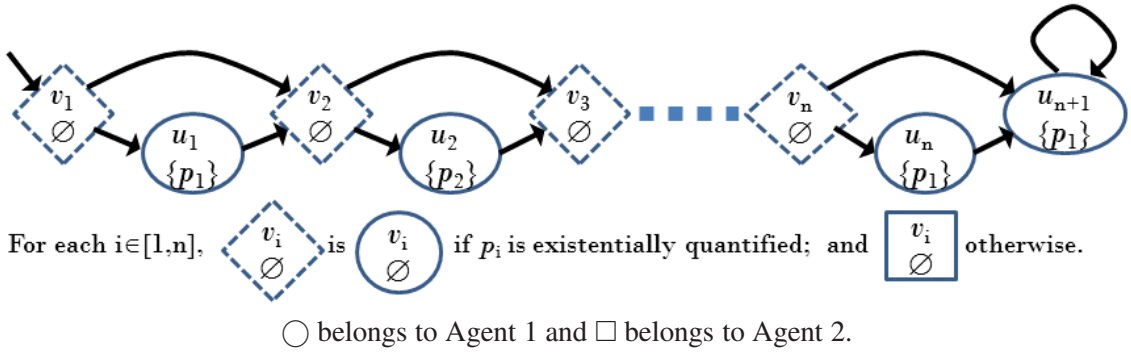


Figure 6: Game graph for the PSPACE-hardness proof of a Boolean formula with n propositions

In the graph, we use oval nodes for states owned by Agent 1 and square nodes for states owned by Agent 2. In each state, we put down its name and the set of atomic propositions that are true at the node. For each atomic proposition $p_i \in P$, we have a corresponding subgraph consisting of nodes v_i and u_i. The subgraph of u_i corresponds to Γ_{p_i} and and that of v_i, u_i together corresponds to Ω_{p_i}.

The design of the graph allows only at most one visit to states $v_1, \ldots, v_n$. For all $i \in [1, n]$, state v_i is owned by Agent 2 if p_i is universally quantified; and owned by Agent 1 otherwise. If p_i is owned by Agent 1, then Agent 1 can choose either (v_i, u_i) or (v_i, v_{i+1}). If p_i is owned by Agent 2, then both choices of Agent 2 at node v_i must yield satisfaction of η.

We construct an ATL$^+$ formula ϕ_η as $\langle 1\rangle \bigwedge_{1\leq i\leq k} \bigvee_{1\leq j\leq h_k} \tau(l_{i,j})$ where $\tau(p) \stackrel{\text{def}}{=} \Diamond p$ and $\tau(\neg p) \stackrel{\text{def}}{=} \Box\neg p$. I.e., ϕ_η is obtained from η by replacing the leading quantifiers in η by $\langle 1\rangle$.

We now show that $\mathcal{G}_\eta \models \phi_\eta$ if, and only if, η is satisfied (i.e., if η is a tautology). The latter is the case if there is a winning strategy for a 'satisfier' in the following game between a 'satisfier' and a 'refuter': following the order of the bound variables, the 'satisfier' and 'refuter' choose the truth value of the existentially and universally bound variables, respectively. When all variables are assigned truth values, the 'satisfier' wins if the CNF formula is satisfied with these values. Otherwise, the refuter wins.

Taking a winning strategy of the satisfier in this game obviously provides a winning strategy for Agent 1 and, vice versa, a winning strategy of Agent 1 in the model checking game can be used as a winning strategy for the 'satisfier' in the satisfaction game. ■

We have an example for the reduction in the proof of Lemma 17.

Example 3 *:* Given $\eta \equiv \exists p\forall q\exists r((p\vee q\vee r)\wedge(\neg p\vee\neg r))$, according to the construction in the proof of Lemma 17, we have the following ATL$^+$ formula: $\phi_\eta \stackrel{\text{def}}{=} \langle 1\rangle(((\Diamond p)\vee(\Diamond q)\vee(\Diamond r))\wedge((\Box\neg p)\vee(\Box\neg r)))$. ■

Following Lemmas 17 and 16, we obtain the complexity of our model-checking problems.

Lemma 18 *The BSIL and ATL$^+$ model-checking problems are PSPACE-complete.* ■

6 Automata for BSIL model-checking

In this section, we discuss a simple encoding of BSIL model-checking in *alternting automatas* on infinite trees [13, 18]. This naturally raises the question why we should study a second approach to BSIL model-checking. The answer is twofold. First, for model-checking itself, it will allow us to establish that the model complexity of BSIL model-checking is polynomial time complete: the problem to decide for a fixed BSIL formula ϕ whether or not a game graph $\mathcal{G}$ is a model of ϕ is P complete. As it is widely believed that models are usually large while specifications are small, a polynomial time bound in the size of the model might be considered more attractive than a PSPACE bound on the complete input. Second, it provides us with the full access of automata based analysis tools, which will allow us to establish a doubly exponential upper bound on the decision problem of whether or not a BSIL formula ϕ is satisfiable.

We start this section by introducing alternating automata, and then continue to encode the model-checking algorithm from the previous section. These automata constructions are then used to establish the inclusion of the model-checking algorithm in PTIME for fixed formulas, while hardness is shown by reducing reachability in AND/OR graphs [12] to model-checking the BSIL formula $\langle 1\rangle\Diamond p$. Beyond establishing PTIME inclusion, we actually show that the problem is fixed parameter tractable: the problem is, for a fixed formula, only quadratic in the size of the model.

6.1 Alternating Automata (AA)

Alternating automata (*AA*) are used to recognized ω-regular tree languages over labeled trees. Let $\mathbb{N}$ denote the set of non-negative integers. Let $\mathbb{D} = [1, d]$ be an interval of $\mathbb{N}$. A $\mathbb{D}$-tree T is a non-empty prefix-closed subset of $\mathbb{D}^*$ (and hence $T \subseteq \mathbb{D}^*$). In T, $\mathbb{D}$ can be interpreted as *directions* from each node in T. Such a T is an ordered tree in the sense that the children of a node in T are naturally ordered according to the directions in $\mathbb{D}$. For example, when $d = 5$, the children of 1212 can be 12122, 12123, and 12125 in order.

An X-labeled tree of Υ is a pair $\langle T, \xi \rangle$, where $\xi : T \to X$ is a function from T to X. We use $\mathbb{B}^+(P)$ to denote the set of positive[3] Boolean combinations of elements in P. A satisfying assignment to a formula in $\mathbb{B}^+(P)$ is a subset of P such that the formula is true if, and only if, all elements in the set is interpreted true.

For the convenience of the readers, we briefly review the definition of AA.

Definition 2 *An alternating automaton (*AA*) A is a tuple $\langle X, U, u_0, \delta, \gamma \rangle$, such that*

- *X is the set of labels of the analysed trees,*
- *U is a finite set of states,*
- *$u_0 \in U$ is an initial state,*
- *$\delta : (U \times X) \mapsto \mathbb{B}^+(\mathbb{D} \times U)$ is a function that maps each pair of a state in U and an input letter in X to a positive Boolean combination of pairs of directions and states, and*
- *$\gamma : U \mapsto \mathbb{N}$ is a valuation function that labels each state with a non-negative integer, called its* priority.

Alternating automata are interpreted over X-labeled trees. A run *of A on an X-labeled tree $\Upsilon = \langle T, \xi \rangle$ is a tree $\langle T', \xi' \rangle$ with $\xi' : T' \mapsto (T \times U)$ with the following inductive restrictions.*

- *$\xi'(\varepsilon) = (\varepsilon, u_0)$. ε is the null sequence.*
- *For every $u \in U$ and sequence $\zeta \in T$ and $\zeta' \in T'$ with $\xi'(\zeta') = (\zeta, u)$ and $\xi(\zeta) = x$, there is a satisfying assignment S of $\delta(u, x)$ such that for every $(i, u') \in S$, there exists a $k \in \mathbb{N}$ with $\zeta' k \in T'$ and $\xi'(\zeta' k) = (\zeta i, u')$.*

As can be seen, the out-degree of a node in T' needs at most $d \cdot |U|$ while that of T is at most d. ■

Intuitively, we want to construct AAs with *states* representing path obligations. The transitions then define in which way these path obligations are sent down the tree. An element (i, u') in the transition formula simply means that the path obligation from u is sent to the child state u' at direction i of the input X-labeled tree.

The priority of a node in a run tree is the priority of the state in its label. The priority of an infinite path is the highest priority taken by infinitely many nodes on the path. A run tree is accepting if the priority of all infinite paths are even, and a tree is accepted if it has an accepting run. The set of trees accepted by such an automaton is called its language and an automaton is called empty if its language is empty.

[3] A Boolean formula is positive if there is no negation in the formula.

An AA is called *nondeterministic* if all functions in the image of δ can be written as a disjunction over conjuncts that contain at most one pair per direction. If they contain only one such disjunct, it is called *deterministic*.

An AA is called a *safety* AA if all its states have the same even priority. For safety AA, the priority function is therefore omitted. An AA is called *weak* if for all its states u and all input letters x, the function $\delta(u, x)$ refers only to states with priorities greater or equal to $\gamma(u)$. It is called a *Büchi* AA if only prioritiy values 1 and 2 are used, i.e., the image of γ is contained in $\{1, 2\}$. Note that weak AA can be rewritten as language equivalent Büchi AA by changing only the priority function from γ to γ' where γ' maps a state u to 2 if $\gamma(u)$ is even, and to 1 otherwise.

6.2 Model Checking with AA's

In this subsection, we relate the algorithms used for model-checking from Section 5 to alternating automata. In order to approach this translation, we start with the simple case where we model check a tree, against a BSIL formula of the form $\langle A\rangle\tau$, where τ does not contain an SQ. This is plausible since we can evaluate those SQs and replace them with auxiliary propositions. In addition, we assume that all strategy decisions are already made. This is also reasonable since we only allow existential SIQs and we forbid negations directly applied to SIQs. Thus we can check the model by checking the existence of S-profiles that fulfill the SQs and SIQs. For convenience, we let $\mathbb{S}$ be the set of strategies that fulfills the SQs and SIQs, if existent.

Like in Section 5, we start with the case that the DNBB formula has only one disjunct. For this case, we model check a tree that has the form of an unravelling of $\mathcal{G}$. To relate this to the automata we have introduced, we assume for the moment that the successors are ordered: a tree node with k successors has a successor in direction 1 through k. Each node is labelled with a triple (a, k, S, P), containing the following information:

- $a \in [1, m]$ shows the owner of the node,
- $k \in \mathbb{D}$ shows the number of successors, and
- $S \subseteq \mathbb{S}$ is the set of strategies, for which we can reach the node. The strategies in S are used just like atomic propositions.
- P is the set of atomic propositions valid in the node.

For such a tree, we check two things for the satisfaction of a BSIL property:

(A1): The labelling of states as reachable is consistent; that is, there are strategies in S such that, for each strategy, exactly the nodes labelled by it are reachable.

(A2): For the respective set of paths described by this labelling, the single disjunct of the DNBB is satisfied.

Note that, by these, we do not check that the form of the tree complies with an unravelling of $\mathcal{G}$. However, the following observation obviously holds.

Lemma 19 *$\mathcal{G}$ is a model of ϕ if, and only if, we can extend the unravelling of $\mathcal{G}$ such that (A1) and (A2) are satisfied.* ■

To test this, we will first show how to construct two AAs that check (A1) and (A2) individually, and then construct a nondeterministic Büchi AA that checks (A1) and (A2) together. Using a nondeterministic AA is attractive because it allows for projecting away the strategies and interpreting $\mathcal{G}$ directly with the resulting AA.

Lemma 20 *We can construct a nondeterministic safety AA $\mathcal{A}_s = \langle X, U_s, u_0^s, \delta^s \rangle$ with $2^{|\mathbb{S}|}$ states that recognises the trees that satisfy (A1).*

Proof : We can simply use $X = [1, m] \times \mathbb{D} \times 2^{\mathbb{S}} \times 2^P$, $U_s = 2^{\mathbb{S}}$, and $u_0^s = \mathbb{S}$. For all $k \leq d$, we have a transition function δ_k^s for the k successor case (of the input labeled tree node) with the following restrictions.

- For an Agent a owning a state and a set of strategies $S \subseteq \mathbb{S}$, we let $S_a \subseteq S$ be the strategy names of strategies for Agent a. We let $\Pi_a^k = \{S_a^1, S_a^2, S_a^3, \ldots, S_a^k\}$ be a disjoint cover of S_a and $S_a^\neg = S - S_a$ be the set of strategies not owned by a.
- We let $\delta_k^s(a, S) = \bigvee_{S_a^1, \ldots, S_a^k \in \Pi_a^k} \bigwedge_{i \in [1,k]} (i, S_a^i \cup S_a^\neg)$. Then we let $\delta_s\big(S, (a, k, S, P)\big) = \delta_k^s(a, S)$ and $\delta_s\big(S, (a, k, S', P)\big) = \mathit{false}$ if $S \neq S'$.

First, the automaton is nondeterministic, and it is easy to see that a run tree must have the same form and the states in its nodes must comply with the strategies in the label of the input tree.

With this observation, the correctness of the construction can be shown by a simple inductive argument. For the induction basis, the initial node is reachable under all strategies, and the run tree is labeled with all strategies. For the induction step, it is easy to show by induction that a node of a (minimal) run tree (and, similarly, the input tree) must have the following property.

- If a state is not labelled with a strategy s_i, then no successor is labeled with s_i.
- If a state is labelled with a strategy s_i owned by a different agent than the node, then all successor states must be labeled with s_i.
- If a state is labelled with a strategy s_i owned by the same agent as the node, then all exactly one successor state is labeled with s_i.

Also, any combination of the labels according to the last rule is possible. This exactly characterize the labelling of reachability for the different strategies. ■

Now we want to construct a weak deterministic AA for property (A2). Let $\mathbb{C}$ denote the set of BP-obligations from property (A2).

Lemma 21 *We can construct a weak deterministic automaton $\mathcal{A}_\wedge = \langle X, U_\wedge, u_0^\wedge, \delta_\wedge, \gamma_\wedge \rangle$ with $2^{|\mathbb{C}|}$ states that recognises the run trees that satisfy (A2).*

Proof : We can simply use $X = [1, m] \times \mathbb{D} \times 2^{\mathbb{S}} \times 2^P$, $U_\wedge = 2^{\mathbb{C}}$, and $u_0^\wedge = \mathbb{C}$. For the transition formulas, we have the following restrictions.

- $\delta_\wedge\big(C,(a,k,S,P)\big) = \mathit{false}$ if there exists a $\Lambda\theta \in C$ with $\Lambda \not\subseteq S$. That is, we require that all transitions can only use strategies that have been passed down from the ancestors.
- $\delta_\wedge\big(C,(a,k,S,P)\big) = \mathit{false}$ if there exists a $\Lambda\theta \in C$ with $\texttt{eval}(P,\theta) = \mathit{false}$. Please recall that $\texttt{eval}(P,\theta)$ converts a θ that is locally violated by P to *false*.
- Otherwise, $\delta_\wedge\big(C,(a,k,S,P)\big) = \bigwedge_{i\in[1,k]}(i,C')$ with $C' = \{\Lambda\texttt{next}(\texttt{eval}(P,\theta)) \mid \Lambda\theta \in C\}$.

Note that $\mathcal{A}_\wedge$ is made deterministic by encoding the choices of strategy passing-down in the input symbol (a,k,S,P). Then we define $\gamma_\wedge$ such that $\gamma_\wedge(C)$ is odd iff C contains an until formula $\phi_1 \mathrm{U} \phi_2$. The correctness of the construction is straightforward. ■

Intersection of $\mathcal{A}_s$ and $\mathcal{A}_\wedge$ can, as usual, be done on the state level.

Corollary 22 *We can construct a nondeterministic weak AA $\mathcal{A}' = \langle [1,m]\times\mathbb{D}\times 2^{\mathbb{S}}\times 2^P, U, u_0, \delta', \gamma\rangle$ with $2^{|\mathbb{S}|+|\mathbb{C}|}$ states that recognises trees that satisfy (A1) and (A2).*

Proof : The new state set U are simply $U_s \times U_\wedge$ and the initial state is $(u_0^s, u_0^\wedge)$. The transition function returns false if either δ_s or $\delta_\wedge$ return false, and are applied independently for the two projections otherwise. Finally, $\gamma(u_s,u_\wedge) = \gamma_\wedge(u_\wedge)$. ■

Corollary 23 *We can construct a nondeterministic weak automaton $\mathcal{A} = \langle [1,m]\times\mathbb{D}\times 2^{\mathbb{S}}\times 2^P, U, u_0, \delta, \gamma\rangle$ with $2^{|\mathbb{S}|+|\mathbb{C}|}$ states that recognises the trees with labelling functions that are projections from the trees that satisfy (A1) and (A2).*

Proof : As usual, this is achieved by choosing $\delta\big(u,(a,k,P)\big) = \bigvee_{S\subseteq\mathbb{S}} \delta'\big(u,(a,k,S,P)\big)$. ■

The above lemmas and corrolaries make it easy to proof our major claim in the section.

Theorem 24 *Model-checking BSIL formulas can be done in time exponential in the BSIL formula and bilinear in the number of states and transitions of the model.*

Proof : Corollary 23 establishes this for PSIL formulas of the form $\phi = \langle A\rangle\tau$, where τ does not contain any SQ. We can extend this to general BSIL state formulas with the following steps.

(1) First, this extends to the case of several disjuncts simply by checking the claim for each disjunct individually.

(2) Second, we can model check this for any state (treating it as the initial state) and subsequently store the result by introducing a fresh atomic proposition p_ϕ, and replace the sub-formula ϕ in the specification by p_ϕ.

(3) Repeating the above two steps until we have reduced the model-checking problem to model-checking a Boolean formula.

This procedure requires the model-checking procedure to be repeated for every SQ as many times as $\mathcal{G}$ has states. (In principle, it would suffice for the topmost SQ to model check it for only the initial state.)

The model-checking algorithm for each state and SQ requires to run up to the number of disjuncts many times the respective consgtruction algorithm of $\mathcal{A}$. We can assume without loss of generality that the turn-based game

structure is properly labeled, as it contains a label for the ownership as well as a label for the atomic propositions, so we only have to add a label for the number of successors and number the directions.

The naïve approach of playing the acceptance game by offering an acceptance player the choice of a disjunct and a rejection player the choice of a direction is obviously too expensive. But note that we can split the decision as follows.

- Let the acceptance player choose the set S from $\delta\big(u,(a,k,P)\big) = \bigvee_{S\subseteq\mathbb{S}} \delta'\big(u,(a,k,S,P)\big)$,
- Let the acceptance and rejection player choose $\delta^k_s(a,S)$ successively by, starting with $I_0 = \mathbb{D}$ and $S_0 = S_a$. Then we use binary search style to iteratively narrow down $I_0, I_1, \ldots$ by repeating the following steps for $i = 0, 1, 2, \ldots$:
  (1) Cut $I_i = [l,u]$ with $m = \lfloor \frac{l+u}{2} \rfloor$ into $I^l_i = [l,m]$ and $I^u_i = [m+1,u]$.
  (2) Let the acceptance player choose disjoint sets L_i and U_i whose union is S_i.
  (3) Let the rejection player choose to continue with either $I_{i+1} = I^l_i$ and $S_{i+1} = L_i$, or with $I_{i+1} = I^u_i$ and $S_{i+1} = U_i$.

  The repetition goes on until $I_i = \{j\}$ is singleton and then execute $\big(j, S_i \cup (S - S_a)\big)$.

That is, by approaching the chosen transition in a logarithmic search we can avoid the overhead. Solving this game is linear in its state space, and this is linear in the number of transition of $\mathcal{G}$. ■

For PTIME hardness, we reduce the reachability checking in an AND/OR graphs [12] to model-checking for the simple BSIL formula $\langle 1\rangle\Diamond p$. For the reduction, it suffices to turn AND nodes to nodes owned by Agent 1 and OR nodes to nodes owned by Agent 2.

Theorem 25 *Model-checking BSIL formulas is PTIME complete in the size of the model.* ■

7 BSIL Satisfiability

In this section, we show that the satisfiability problem of BSIL is 2EXPTIME complete. The upper bound can be inferred by the simulation theorem [], while hardness is a consequence of the inclusion of CTL$^+$.

In the first step, we provide an AA for checking the consistency of a labelling as it is constructed in the model-checking approach in the previous section. This labelling contains explicit information about whether or not a BSIL SQ formula is satisfiable. In the following we assume without loss of generality that every state in the turn based game is reachable from the initial state without mentioning this explicitly. (Note that unreachable states have no impact on the correctness and can simply be removed.)

Lemma 26 *We can build an AA $\mathcal{B}$ which is exponential in the size of a BSIL formula χ and accepts a fully labelled turn-based game iff the labelling is consistent and the turn-based game structure is a model of χ.*

Proof : For each SQ subformula ϕ of χ, we can build a weak nondeterministic AA $\mathcal{A}_\phi$ that consists of the nondeterministic AA $\mathcal{A}_\eta$ for each disjunct η in the DNBB formula of ϕ (where we assume, without loss of generality, that

their states are disjoint and $\mathcal{A}_\eta$ has initial state u_η) plus a fresh initial state u_0 with priority 0. The transition function for u_0 is simply $\delta(u_0, x) = \bigvee_\eta \delta(u_\eta, x)$, that is, in the first step one arbitrary individual automata is entered and never left again.

Likewise, we can build a weak AA $\mathcal{D}_\phi$ that accepts the complement language of $\mathcal{A}_\phi$ by dualising it. Let K denote the set of subformulas of χ that start with an SQ. Assume, without loss of generality, that the states of the individual $\mathcal{A}_\phi$ and $\mathcal{D}_\phi$ are disjoint and that the initial states of $\mathcal{A}_\phi$ and $\mathcal{D}_\phi$ are u_ϕ and $\overline{u}_\phi$, respectively. Then we can build $\mathcal{B}$ by adding two fresh states, a state u and the initial state u_0 (both with priority 0) and define the transition formula as follows.

- $\delta\big(u_0, (a, k, P)\big) =$ *false* if $P \not\models \chi$ and
- $\delta\big(u_0, (a, k, P)\big) = \delta\big(u, (a, k, P)\big)$ if $P \models \chi$,
- $\delta\big(u, (a, k, P)\big) = \bigwedge\limits_{i \in [1,k]} (i, u) \wedge \bigwedge_{\phi \in K, p_\phi \in P} \delta\big(u_\eta, (a, k, P)\big) \wedge \bigwedge_{\phi \in K, p_\phi \notin P} \delta\big(\overline{u}_\eta, (a, k, P)\big)$.

For the states of the individual $\mathcal{A}_\phi$ and $\mathcal{D}_\phi$, their transition and priority function is used. ■

Theorem 27 *The satisfiability of a BSIL formula ϕ can be checked in time doubly exponential in the size of φ. If φ is satisfiable, then a model can be constructed in time doubly exponential in φ.*

Proof : The main difference to the model-checking case is that we cannot infer a sufficient branching degree from the model. We therefore approach in two steps: we assume that we know a sufficient branching degree, and discuss a synthesis algorithm for it.

The first observation is that, for states in a $\mathcal{D}_\phi$, we can use any subtree as this automaton shows that something holds for all strategies. The reduction to a subtree makes the property easier to satisfy. The second observation is that a winning strategy for the acceptance player can be assumed to be memoryless. Thus, for each state of an AA $\mathcal{A}_\phi$ and each state of a tree accepted, strategies from the respective S_a (from the proof for Lemma 20) are sent to at most $|S_a| \leq |\mathbb{S}|$ directions. But successors of nodes to whom from no state a non-empty subset of S_a is sent can be pruned, provided at least one successor remains. This gives a bound on the number of directions needed, which is bilinear in the number of states all $\mathcal{A}_\phi$ put together and the maximal number of strategies. (The latter can be estimated by the number of SIQs bound by any SQ plus one times the number of agents.) Thus, the number of directions can be restricted to a number exponential in χ.

Having established this bound of directions, we can build a language equivalent nondeterministic Büchi AA in time exponential in $\mathcal{B}$ (for the given k-bounded branching degree), whose emptiness can be checked in polynomial time. ■

remark

Strictly speaking, a specification alone does not reveal the number of agents. There are two interpretations possible: either one requires the agents to be explicitly named, or one leaves the set of agents open. In the latter case, however,

all agents that are not named explicitly co-operate, and their strategy is never bound in any conjunct of a DNBB. Hence, they can be collapsed to a single agent without changing the semantics.

To obtain hardness we use draw from the 2EXPTIME-completeness of CTL$^+$. It is easy to see that CTL$^+$ is the sub-logic of BSIL, where $\langle A \rangle$ is restricted to $\langle \emptyset \rangle$ and $\langle +A \rangle$ operators are restricted to $\langle +\emptyset \rangle$ operators and always bind a temporal operator. Together with the previous theorem, the 2EXPTIME hardness of CTL$^+$ [] provides us with:

Corollary 28 *Checking the realisability of a BSIL formula φ 2EXPTIME-complete in the size of φ.*

8 Experiment

8.1 Implementation

We implemented a semi-symbolic model checker of BSIL with **REDLIB** [**?**,**?**,**?**,**?**,**?**] which is a free library for symbolic model-checking based on decision diagrams. **REDLIB** supports symbolic pre-condition and post-condition calculation of discrete transitions. Our model-checker starts from a symbolic representataion of the initial condition. Then it repeatedly applies the post-condition procedure of **REDLIB** to explore the symbolic state representations in the computation tree. Our model-checker use the result in section 5 to bound the exploration depth of tree.

8.2 Benchmarks and their experiment report

Then we experimented with two parameterized benchmarks. The first is the prisoners' dilemma described in Example 1 with the number of prisoners as a parameter. Then we applied the model checker to check whether the model satisfies formula (A) and (B) in the introduction. The time and memory usage of the model checker is reported in Table 2.

Table 2: Experiment data for the prisoners' dilemma model

#prisoners	Formula (A)		Formula (B)	
	Time	Mem	Time	Mem
2	0.50s	72M	0.51s	71M
3	0.92s	75M	0.88s	75M
4	1.23s	83M	1.14s	80M
5	3.19s	146M	2.90s	140M
6	6.71s	388M	6.52s	361M
7	25.10s	1043M	23.11s	979M

s: seconds (computation time);
M: megabytes (memory usage);

The experiment is conducted on a PC with Intel i7-2600k 3.4GHZ CPU and 8G RAM running Ubuntu 12.04.

The second benchmark is for campaign strategies of 2 political parties, Party 1 and Party 2, for seats in the congress. There are several parameters in our model, the number of seats in the congress, the amount of budget of

each party in a round, and the number of candidates of each party in a round. The game is played by the chiefs of the parties, Chief 1 (a gentleman) and Chief 2 (a lady), who decide the amount of support that a candidate receive in a round. Candidates with more budget in a round will be elected in the round. If two or more candidates get the same amount of support, the winner will be decided randomly.

In our model, in each round, chief 1 first decides his budget allotment, then chief 2 decides her budget allotment, and then the election result is determined. For example, with 3 dollars in his budget for two candidates, the strategy of a Chief can be written as $(x, y) \in \{(3,0),(2,1),(1,2),(0,3)\}$ where x and y respectively denote the support (in dollars) alloted to the first and the second candidates in his party. If Chief 1 uses strategy $(2,1)$ and Chief 2 uses strategy $(0,2)$ (Assume she has less budget.), the result of the round is that the two parties both win one seat in the round. On the other hand, if Chief 2 chooses strategy $(1,1)$ instead, then there is no strategy for Chief 1 to win more seats for his party.

We use the following propositions and predicates to write specifications. For each $i \in [1,2]$ and $j \in [1,c]$ where c is the number of candidates, $\mathit{elected}_{i,j}$ means the j'th candidate of party i is elected now. Then we can use propositions $\mathit{elected}_{i,j}$ to construct a predicate win_i that means party i wins more seats than party $2-i$ in the round. Assume that the first candidate in each party is the chief. Then the following BSIL formula specifies that Chief 1 has a strategy to make sure candidate 1 to be elected and in the same time allow the opponent party to win more seats in every round.

$$\langle 1\rangle((\langle +\rangle \Diamond \mathit{elected}_{1,1}) \wedge (\langle +2\rangle \Box \mathit{win}_2)) \tag{C}$$

Similarly, the following formula specifies that the chief of party 1 has a strategy to assure that his party can win more seats than his opponent party while allowing party two to assure the winning of a candidate.

$$\langle 1\rangle((\langle +\rangle \Diamond \mathit{win}_1) \wedge (\langle +2\rangle \Diamond \mathit{elected}_{2,1})) \tag{D}$$

With different values of budget and candidate count, we may use our model-checker to analyze the strategy for achieving various goals. The time and memory usage data for checking formula (C) and (D) is in Table 3.

The experiment data shows that our tool can be used in analyzing strategies for winning games. It would be interesting to see how general and specific performance improvement techniques could be designed in the future fro practical industry projects.

9 Conclusion

BSIL can be useful in describing close collaboration among strategies of agents in a multi-agent system. Although it can express properties that ATL*, GL, and AMC cannot, its model-checking problem incurs a much lower complexity. Future work in this direction may include further extension to BSIL and better efficient model-checking algorithms.

Table 3: Experiment Data Election

Parameters					Formula (C)			Formula (D)		
c_1	b_1	c_2	b_2	s	Result	Time	Mem	Result	Time	Mem
2	2	2	4	1	UNSAT	0.52s	61M	UNSAT	0.53s	61M
2	4	2	4	2	UNSAT	1.20s	75M	UNSAT	1.15s	75M
2	4	2	4	3	SAT	0.73s	75M	UNSAT	1.14s	75M
3	6	3	6	3	SAT	1.51s	211M	UNSAT	0.93s	211M
3	9	3	6	3	UNSAT	9.14s	837M	SAT	5.22s	681M
3	6	3	9	3	SAT	8.74s	502M	UNSAT	8.49s	502M
3	9	3	9	4	SAT	158s	6631M	UNSAT	93.17s	5082M

c_1: # candidates of party 1
b_1: total budget in a round of party 1
c_1: # candidates of party 2
b_1: total budget in a round of party 2
s: # of seats in the congress

s: seconds (computation time)
M: megabytes (memory usage)

The experiment is conducted on a PC with Intel i7-2600k 3.4GHZ CPU and 8G RAM running Ubuntu 12.04.

Acknowledgment

The authors would like to thank Moshe Vardi and the anonymous reviewers of CONCUR 2011 for their valuable comments and suggestions.

References

[1]

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

[3] C. Baier, T. Brázdil, M. Größer, and A. Kucera. Stochastic game logic. In *QEST*, pages 227–236. IEEE Computer Society, 2007.

[4] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[5] K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. *Information and Computation*, 208:677–693, 2010.

[6] A. D. Costa, F. Laroussinie, and N. Markey. Atl with strategy contexts: Expressiveness and model checking. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 120–132. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

[7] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs as fixpoints. In *7th Colloquium on Automata, Language, and Programming*, volume LNCS 85. Springer-Verlag, 1980.

[8] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1–24, Feb. 1985.

[9] E. A. Emerson and J. Y. Halpern. 'sometimes' and 'not never' revisited: On branching versus linear time temporal logic. *Journal of ACM*, 33(1):151–178, Jan. 1986.

[10] E. A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Company, 1979.

[12] N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):65–72, 1981.

[13] D. Kirsten. Alternating tree automata and parity games. In E. Gradel, W. Thomas, and T. Wilke, editors, *Automata, Logics, and Infinite Games*, volume LNCS 2500, pages 3?V–21. Springer, 2002.

[14] L. Lamport. Sometimes is sometimes "not never"-on the temporal logic of programs. In *7th Annual ACM Symp. on Principles of Programming Languages*, pages 174–185, 1980.

[15] F. Mogavero, A. Murano, and M. Y. Vardi. Reasoning about strategies. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 133–144. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

[16] A. Sistla and E. Clarke. The complexity of propositional linear temporal logics. *JACM*, 32, 1985.

[17] L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. MIT, 1974.

[18] J. Zappe. Modal μ-calculus and alternating tree automata. In E. Gradel, W. Thomas, and T. Wilke, editors, *Automata, Logics, and Infinite Games*, volume LNCS 2500, pages 3?V–21. Springer, 2002.