A Temporal Logic for the Interaction of Strategies[⋆,⋆⋆,⋆⋆⋆]

Farn Wang[1,2], Chung-Hao Huang[2], and Fang Yu[3]

[1] Dept. of Electrical Engineering, National Taiwan University
[2] Graduate Institute of Electronic Engineering, National Taiwan University
[3] Dept. of Management Information Systems, National Chengchi University

Abstract. We propose an extension to *ATL* (*alternating-time logic*), called *BSIL* (*basic strategy-interaction logic*), for the specification of interaction among the strategies of agents in a multi-agent system. BSIL allows for the specifications of one system strategy that can cooperate with several strategies of the environment for different requirements. We argue that such properties are important in practice and rigorously show that such properties are not expressible in *ATL**, *GL* (*game logic*), and *AMC* (*alternating μ-calculus*). Specifically, we show that BSIL is 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 show that the model-checking problem of BSIL is PSPACE-complete and is of lower complexity than those of ATL*, GL, AMC, and the general strategy logics. This may imply that BSIL can be useful in closing the gap between real-world projects and the game-theoretical results. We then show the plausibility and feasibility of our techniques by reporting our implementation and experiment with our PSPACE model-checking algorithm for BSIL. Finally, we discuss an extension of BSIL.

Keywords: games, turn-based, logic, model-checking, expressiveness.

1 Introduction

The specification and verification of open systems focuses on the design of system interfaces that allow for the fulfillment of various functions and prevent other bad behaviors from happening. The theoretical challenges in designing such systems have drawn the attention of researchers in game theory. From the perspective of game theory, the design problem of such open systems can be modeled as a multi-agent game. Some players represent the system while other players represent the environment (or the users). The system wins the game in an execution (or a *play* in the jargon of game theory) if all the system specifications are fulfilled along the execution. The goal of the system design, from the perspective of game theory, is to design a computable strategy of the system

[⋆] The work is partially supported by NSC (Grant NSC 97-2221-E-002-129-MY3, Taiwan, ROC) and by Research Center for Information Technology Innovation, Academia Sinica, Taiwan, ROC.

[⋆⋆] The authors would like to thank Prof. Moshe Vardi for his valuable suggestions.

[⋆⋆⋆] A full version of the work with all the lemma proofs can be found in `http://cc.ee.ntu.edu.tw/~farn`.

that enforces all the system specifications. Such a strategy is called a *winning strategy* for the system.

At the moment, there are various game-theoretical languages, including *ATL* (*alternating-time logic*), *ATL**, *AMC* (*alternating μ-calculus*), *GL* (*game logic*) [1], and *SL* (*strategy logics*) [4, 3, 6], for the specification of open systems. Each language also comes with a verification algorithm that helps in deciding whether a winning strategy for the system exists. However, there is a gap between the need of the industry and the technology to offer from the 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 [1] allow us to specify that some players together have a strategy to fulfill something. This is far from what the industry need in specification. Consider the following example of a banking system.

Example 1. **Banking system** A bank needs to specify that their banking system, embodied as a system strategy, allows 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 the client's strategies for good behaviors while prevent client's strategy for bad behaviors from damaging the system. We actually prove in this work that *ATL* (*alternating-time logic*), *ATL**, *AMC* (*alternating μ-calculus*), and *GL* (*game logic*) [1] do not support the specifications in this regard. For example, it is not possible to specify with those languages that the system strategies used both in a withdrawal transaction and in a deposit transaction must be the same. As a result, the verification algorithms of those languages actually do not help as much as we wish in verifying real-world open systems. ■

To solve the expressiveness problem in the above example, strategy logics were proposed in [4, 3, 6] 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 retrospection, the specification problem of the above-mentioned properties has a deep root in game theory. Consider a game among 3 prisoners initially in jail.

Example 2. **Prisoners' dilemma** A prisoner may deny charges or may cooperate with the police. If all deny, they are all acquitted of the charges. If more than one choose to cooperate, all will stay in jail. If all but one deny, then all will be in jail except the sole cooperating one will be a dirty witness and be acquitted. We may want to specify that the three prisoners may collaborate with each other, will all deny, and will not be in jail. Let j_a be the proposition for prisoner a in jail. This can be expressed respectively in Alur, Henzinger, and Kupferman's ATL, ATL*, GL, and AMC [1] as follows.[1]

$$\begin{array}{ll} \text{ATL, ATL}^*\text{:} & \langle\{1,2,3\}\rangle \bigwedge_{a\in[1,3]} \Diamond\neg j_a \\ \text{GL:} & \exists\!\!\!\exists\{1,2,3\}. \bigwedge_{a\in[1,3]} \forall\Diamond\neg j_a \\ \text{AMC:} & \mathbf{lfp}X.\langle\{1,2,3\}\rangle \bigcirc \bigwedge_{a\in[1,3]}(X \vee \neg j_a) \end{array}$$

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

Here "$\langle\{1,2,3\}\rangle$" and "$\exists\{1,2,3\}$" 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. "**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 history in game theory. If we recall example 1, it is easy to see the similarity and link between this example and the banking system specification problem. This observation may suggest that finding a language with an appropriate balance between the expressiveness and the verification complexity is still a central challenge yet to be overcome. ■

To meet the challenge, we propose an extension to ATL, called *BSIL* (*basic strategy-interaction logic*), by introducing a new modal operator called *strategy interaction quantifier* (*SIQ*). In the following, we use several examples in the prisoners' dilemma to explain BSIL. A formula for the property in example 2 follows.

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

Here "$\langle+\{3\}\rangle$" is an existential SIQ on strategies of prisoner 3 for collaborating with the strategies of prisoners declared in the parent formula. Similarly, "$\langle+\emptyset\rangle$" means that no collaboration of any prisoner is needed. 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.

If prisoner 1 really hates the others, he can always cooperate with the police, make sure that prisoners 2 and 3 will be in jail, and let them decide whether he will be in jail. This property can be expressed in BSIL as follows.

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

At last, we can also use BSIL to express the deterministic Nash equilibrium, a scenario in which a unilateral change of actions by a prisoner does not improve her/his payoff. In the example, an equilibrium is that all prisoners keep on cooperating with the police. Such an equilibrium results in the following property in BSIL.

$$\langle\{1,2,3\}\rangle((\textstyle\bigwedge_{a\in[1,3]}\langle+\emptyset\rangle\Box j_a) \wedge \textstyle\bigwedge_{a\in[1,3]}\langle+\{a\}\rangle\Box\textstyle\bigwedge_{b\in[1,3];b\neq a} j_b)$$

Note that we let the inner quantifications "$\langle+\{1\}\rangle$", "$\langle+\{2\}\rangle$", and "$\langle+\{3\}\rangle$" to overrule the strategy binding by "$\langle\{1,2,3\}\rangle$."

In this work, we establish that BSIL is incomparable with ATL*, GL, and AMC in expressiveness. Although strategy logics proposed by Costa, Laroussinie, Markey ite-CLM10, Chatterjee, Henzinger, Piterman [3], Mogavero, Murano, and Vardi [6] 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 game graphs. This may imply that BSIL could be a better balance between expressiveness and verification efficiency than ATL*, GL, AMC [1], and SL [3,6]. Moreover, the

[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).

deterministic Nash equilibria expressed in BSIL in the above may also imply that BSIL could be a valuable and useful subclass of strategy logics [3,6].[3]

We also establish some other properties of BSIL. We show that the strategies for BSIL properties against turn-based games need be memoryful. We prove that the BSIL model-checking problem is PSPACE-complete. We have also implemented our model-checking algorithm and carried out experiments. Finally, by lifting the restriction that no SIQ may appear as a topmost SQ in a path subformula, we can extend BSIL to *strategy interaction logic* (*SIL*) by allowing the SIQs to be used directly after temporal modal operators. For example, we may have the following SIL formula.

$$\langle\{1\}\rangle\Box((p \rightarrow \langle +\{2\}\rangle \bigcirc q) \wedge (\neg p \rightarrow [+\{2\}] \bigcirc \neg q))$$

SIL has the expressiveness for the interaction of a player's stratregy with infinitely many strategies used by players at different states along a play. We also show that SIL is strictly more expressive than ATL* and its model-checking problem is doubly exponential time hard.

Here is our presentation plan. Section 2 explains turn-based game graphs for the description of multi-agent systems and presents BSIL. Section 3 shows that BSIL is more expressive than ATL [1] but not comparable with ATL*, AMC, and GL [1] in expressiveness. Section 4 shows that BSIL model-checking problem needs memoryful strategies and is PSPACE-hard. Section 5 presents a PSPACE algorithm for BSIL model-checking and establishes that BSIL model-checking problem is PSPACE-complete. Section 6 extends BSIL to SIL. Section 7 is the conclusion.

2 System Models and BSIL

2.1 Turn-Based Game Graphs

A *turn-based* game is played by many agents. Assume that the number of agents is m and we index the agents with integers 1 through m. It is formally presented as a tuple $A = \langle m, Q, q_0, \omega, P, \lambda, R\rangle$ with the following restrictions. m is the number of agents in the game. Q is a finite set of states. $q_0 \in Q$ is the *initial state* of A. $\omega : Q \mapsto [1, m]$ is a function that specifies the owner of each state. Only the owner of a state makes choice at the state. P is a finite set of atomic propositions. $\lambda : Q \mapsto 2^P$ is a proposition labeling function. $R \subseteq Q \times Q$ is the set of transitions. In figure 1, we have the graphical representation of a turn-based game graph with initial state v. The ovals and squares represent states while the arcs represent state transitions. We also put down the λ values inside the corresponding states.

A state predicate of P is a Boolean combination of elements in P. We let $SP(P)$ be the set of state predicates of P. The satisfaction of a state predicate η at a state q, in symbols $q \models \eta$, is defined in a standard way.

For convenience, given a game graph $A = \langle m, Q, q_0, \omega, P, \lambda, R\rangle$, we denote $m, Q, q_0, \omega, P, \lambda$, and R respectively as $m_A, Q_A, q_{0A}, \omega_A, P_A, \lambda_A$, and R_A.

[3] Another work worthy of mentioning is the stochastic game logic (SGL) by Baier, Brázdil Gröser, and Kucera [2] with limited expressiveness for strategy interaction. However, for memoryful strategies, the model-checking problem of SGL is undecidable.

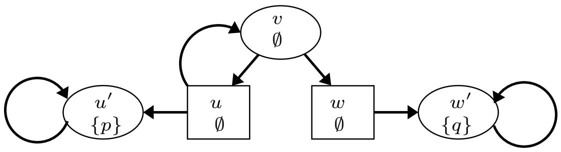

Fig. 1. A turn-based game graph

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 let $|\rho| = n+1$. For convenience, we use $\mathit{last}(\rho)$ to denote the last state in ρ, i.e., $\rho(|\rho|-1)$.

For an agent $a \in [1,m]$, a *strategy* σ for a is a function from Q_A^* to Q_A such that for every $\rho \in Q_A^*$, $\sigma(\rho) \in Q_A$ with $(\mathit{last}(\rho),\sigma(\rho)) \in R_A$. A set of agents is called an *agency*. A *congregate strategy* (or *C-strategy*) Σ of an agency $M \subseteq [1,m]$ is a partial function from $[1,m]$ to the set of strategies such that for every $a \in [1,m]$, $a \in M$ iff[4] $\Sigma(a)$ is defined. The composition of two C-strategies 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.

Later, we will use composition of C-strategies 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 $k \in [0,\infty)$, $\omega(\rho(k)) = a$ implies $\sigma(\rho[0..k]) = \rho(k+1)$. The play is compatible with a C-strategy Σ of agency M iff for every $a \in M$, the play is compatible with $\Sigma(a)$ of a.

2.2 BSIL Syntax

For a turn-based game graph A 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 traces. A BSIL formula ϕ is constructed with the following three syntax rules, respectively for state formula ϕ, tree formula θ, and path formula ψ.

$$\phi ::= p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \langle M\rangle\theta \mid \langle M\rangle\psi$$
$$\theta ::= \neg\theta_1 \mid \theta_1 \vee \theta_2 \mid \langle +M\rangle\theta_1 \mid \langle +M\rangle\psi$$
$$\psi ::= \bigcirc\phi \mid \phi_1 \mathrm{U} \phi_2 \mid \phi_1 \mathrm{W} \phi_2$$

Here p is an atomic proposition in P_A. M is a subset of $[1,m]$. $\langle M\rangle\psi$ means that there exist strategies of the agents in M that make all plays satisfy ψ. Formulas $\langle +L\rangle\theta_1$ and $\langle +L\rangle\psi$ must happen as subformulas of a state formula $\langle M\rangle\theta$. Intuitively, they mean that there exist strategies of L that work with the strategies bound to $\langle M\rangle\theta$ to make θ_1 and ψ true respectively.

[4] "*iff*" is a shorthand for "*if and only if.*"

Formulas ϕ are called *BSIL formulas*. 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 the state. For convenience, we also have the following shorthands.

$$\begin{array}{rlrl}
\mathit{true} & \equiv p \vee (\neg p) & \mathit{false} & \equiv \neg \mathit{true} \\
\phi_1 \wedge \phi_2 & \equiv \neg((\neg\phi_1) \vee (\neg\phi_2)) & \phi_1 \Rightarrow \phi_2 & \equiv (\neg\phi_1) \vee \phi_2 \\
\Diamond\phi_1 & \equiv \mathit{true}\,\mathbf{U}\phi_1 & \Box\phi_1 & \equiv \phi_1\mathbf{W}\mathit{false} \\
[M] \bigcirc \phi_1 & \equiv \neg\langle M\rangle \bigcirc \neg\phi_1 & [+M] \bigcirc \phi_1 & \equiv \neg\langle +M\rangle \bigcirc \neg\phi_1 \\
[M]\phi_1\mathbf{U}\phi_2 & \equiv \neg\langle M\rangle\neg\phi_2\mathbf{W}\neg(\phi_1 \vee \phi_2) & [+M]\phi_1\mathbf{U}\phi_2 & \equiv \neg\langle_{M}\rangle\neg\phi_2\mathbf{W}\neg(\phi_1 \vee \phi_2) \\
[M]\phi_1\mathbf{W}\phi_2 & \equiv \neg\langle M\rangle\neg\phi_2\mathbf{U}\neg(\phi_1 \vee \phi_2) & [+M]\phi_1\mathbf{W}\phi_2 & \equiv \neg\langle_{M}\rangle\neg\phi_2\mathbf{U}\neg(\phi_1 \vee \phi_2)
\end{array}$$

Operators $\langle \ldots \rangle$ and $[\ldots]$ are all *SQ*s. Operators $\langle + \ldots \rangle$ and $[+ \ldots]$ are *SIQ*s. All BSIL formulas are *well-formed* since all their SIQs occur inside an SQ beginning with $\langle M \rangle$ for some M.

2.3 BSIL Semantics

BSIL subformulas are interpreted with respect to C-strategies. A state or a tree formula ϕ is satisfied at a state q with C-strategy Σ, in symbols $A, q \models_\Sigma \phi$, if and only if the following inductive constraints are satisfied.

- $A, q \models_\Sigma p$ iff $p \in \lambda(q)$.
- For state or tree formula ϕ_1, $A, q \models_\Sigma \neg\phi_1$ iff $A, q \models_\Sigma \phi_1$ is false.
- For state or tree formulas ϕ_1 and ϕ_2, $A, q \models_\Sigma \phi_1 \vee \phi_2$ iff either $A, q \models_\Sigma \phi_1$ or $A, q \models_\Sigma \phi_2$.
- $A, q \models_\Sigma \langle M\rangle\theta$ iff there exists a C-strategy Π of M with $A, q \models_\Pi \theta$.
- $A, q \models_\Sigma \langle +M\rangle\theta$ iff there exists a C-strategy Π of M with $A, q \models_{\Sigma\circ\Pi} \theta$. Here function composition $\Sigma \circ \Pi$ models inheritance of strategy bindings Σ from ancestor formulas.
- $A, q \models_\Sigma \langle M\rangle\psi$ iff there exists a C-strategy Π of M such that for all plays ρ from q compatible with Π, $\rho \models_\Pi \psi$ which means that ρ satisfies path formula ψ with C-strategy Π.
- $A, q \models_\Sigma \langle +M\rangle\psi$ iff there exists a C-strategy Π of M such that for all plays ρ from q compatible with $\Sigma \circ \Pi$, $\rho \models_{\Sigma\circ\Pi} \psi$.

Note that we also let a play ρ satisfy a path formula ψ with the inheritance of a C-strategy. This is in fact not necessary for BSIL semantics since all such inheritance will be overruled by SQs immediately following the temporal modal operators. However, this is necessary when we later extend BSIL by allowing for the inhertance of strategies across the temporal modal operators.

A play ρ satisfies a path formula ψ with C-strategy Σ, in symbols $\rho \models_\Sigma \psi$, iff the following restrictions hold.

- $\rho \models_\Sigma \bigcirc\phi_1$ iff $A, \rho(1) \models_\Sigma \phi_1$.
- $\rho \models_\Sigma \phi_1\mathbf{U}\phi_2$ iff there exists an $h \geq 0$ with $A, \rho(h) \models_\Sigma \phi_2$ and for all $j \in [0, h)$, $A, \rho(j) \models_\Sigma \phi_1$.
- $\rho \models_\Sigma \phi_1\mathbf{W}\phi_2$ iff either $\rho \models_\Sigma \phi_1\mathbf{U}\phi_2$ or for all $h \geq 0$, $A, \rho(h) \models_\Sigma \phi_1$.

For convenience, we let $\perp$ be a null C-strategy, i.e., a function that is undefined on everything. If ϕ_1 is a BSIL (state) formula and $A, q \models_\perp \phi_1$, then we may simply write $A, q \models \phi_1$. If $A, q_0 \models \phi_1$, then we also write $A \models \phi_1$.

3 Expressiveness of BSIL

In this section, we establish that BSIL is incomparable with ATL*, AMC, GL [1] in expressiveness. It is easy to see that BSIL is a super-class of ATL. Thus we have the following lemma.

Lemma 1. *BSIL is at least as expressive as ATL.* ■

Lemmas 2 and 3 establish that ATL* and BSIL are incomparable.

Lemma 2. *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 $A_0, \ldots, A_k, \ldots$ and $B_0, \ldots, B_k, \ldots$ of game graphs such that no ATL* formula with k SQs can distinguish A_k and B_k. ■

Lemmas 1 and 2 together establish that ATL is strictly less expressive than BSIL.

Lemma 3. *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 [1]. ■

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

Lemma 4. *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 2. ■

Lemma 5. *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 [1] that $\exists\!\!\exists\{1\}.((\exists\Box p) \wedge \exists\Box q)$ is not equivalent to any ATL* formula. ■

To establish that AMC is not as expressive as BSIL, we basically follow the proof style for lemma 2 and use the same two families of game graphs. The statement of the lemma requires notations for proposition variables and other details in AMC.

Lemma 6. *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 : The proof is similar to the one for lemma 2. ■

By the same argument in [1], for one-agent game, BSIL coincides with CTL and is not as expressive as AMC.

Lemma 7. *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 2, 4, and 6 is that the path modal formulas in the lemmas can be changed independently to $\Diamond p$ and $\Diamond q$ 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.

4 Some Properties of BSIL Model Checking Problem

In this section, we first show that a strategy synthesizing a BSIL formula needs memory. A strategy σ is *memory-less* iff for every two play prefixes ρ and ρ', $\mathit{last}(\rho) = \mathit{last}(\rho')$ implies $\sigma(\rho) = \sigma(\rho')$.

Lemma 8. *There is a BSIL model-checking problem instance that cannot be satisfied with a memory-less strategy of an agent.*
Proof : Consider the 2-agent game graph in figure 1. Again we use ovals to represent those nodes owned by agent 1 and squares to represent those by agent 2. We want to check property $\langle\{1\}\rangle((\langle+\{2\}\rangle\Diamond p) \wedge (\langle+\{2\}\rangle\Diamond q))$. It is clear that no memory-less strategy of agent 1 satisfies this property. However, a strategy of agent 1 that chooses arc (v, u) at least once and eventually chooses arc (v, w) satisfies the BSIL property. ■

In the following, we establish the complexity lower-bound of the model-checking problem of BSIL formulas. This is done by reducing the QBF (quantified Boolean formula) satisfiability problem [5] to BSIL model-checking problem of 3-agent game graphs. We assume a QBF property $\eta \equiv \mathcal{Q}_1 p_1 \ldots \mathcal{Q}_l p_l (C_1 \wedge C_2 \wedge \ldots \wedge C_n)$ with a set $P = \{p_1, \ldots, p_l\}$ of atomic propositions and the following restrictions.

- For each $k \in [1, l]$, $\mathcal{Q}_k$ 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 a BSIL formula for a traversal requirement on the game graph. The game graph has some special components corresponding to the truth values of each atomic proposition. Suppose that Γ_p and $\Gamma_{\neg p}$ respectively represent the subgraphs for the truth and falsehood 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 QBF formula actually can be interpreted as a requirement for covering those Γ_p's and $\Gamma_{\neg p}$'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*
- *one of $\Gamma_{\neg p}$ and $\Gamma_{\neg r}$ is covered.*

Of course, we have to make sure that Γ_p and $\Gamma_{\neg p}$ cannot be covered in the same play for each p. The details of constructing those Γ_p's, $\Gamma_{\neg p}$'s, and Ω_p's can be found in the proof of the following lemma that establishes the PSPACE complexity lower-bound.

Lemma 9. *BSIL model-checking problem for turn-based game graphs is PSPACE-hard.* ■

5 BSIL Model-Checking Algorithm

There are two restrictions on BSIL formulas that enable us of the design of a PSPACE model-checking algorithm. Firstly, similar to ATL and CTL, each path modal operator must occur immediately after an SQ. Secondly, an SIQ cannot occur as a topmost SQ in a path formula. These two restrictions together suggest that as in the model-checking algorithms of ATL [1], we can evaluate the proper subformulas of path modal formulas independently and then treat them as auxiliary propositions. Moreover, as in the evaluation of $\langle\{\ldots\}\rangle\Diamond$-formulas in ATL model-checking, if a $\Diamond$-formula can be enforced with a strategy, it can be enforced in a finite number of steps along every play compatible with the strategy in a computation tree. Once the bound b of this "finite number" of steps is determined, then we can enumerate all the strategies embedded in the computation tree up to depth b and try to find one that enforces a BSIL formula.

One thing in exploring a computation tree for BSIL different from that for ATL [1] is that we have to consider the interaction of strategies. For example, we may have a subformula $\langle\{1\}\rangle((\langle+\{2\}\rangle\Diamond p)\wedge\langle+\{2\}\rangle\Box q)$ to enforce. Then in exploring the computation tree, we may follow two strategies of agent 2, one to enforce $\Diamond p$ and the other to enforce $\Box q$, that always make the same decision until we reach a tree node v owned by agent 2. This can be conceptualized as passing the obligations of $\Diamond p$ and $\Box q$ 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 strategies declared in a BSIL formula.

Then in subsection 5.2, we present our algorithm into two parts, one for the checking of BSIL state formulas and the other for that of 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.

5.1 Computing Path Obligations Passed Down Computation Tree

We need some special techniques in checking tree formulas. We adopt the concept of strategy variables from [3,6]. Given $\{a_1,\ldots,a_n\}\subseteq[1,m]$, we use $\{a_1\mapsto s_1,\ldots,a_n\mapsto s_n\}$ to denote the *strategy variable binding* (*SV-binding* for short) that binds agents $a_1,\ldots,a_n$ respectively to strategy variables $s_1,\ldots,s_n$. Given an SV-binding B, $B\circ\{a_1\mapsto s_1,\ldots,a_n\mapsto s_n\}$ is the SV-binding that is identical to B except that agent a_i is bound to s_i for every $i\in[1,n]$. Given an agency $M\subseteq[1,m]$, $B^{\neg M}$ is defined as the subset of B defined on $[1,m]-M$. Specifically, $B^{\neg M}$ is $\{a\mapsto s\mid a\mapsto s\in B, a\notin M\}$. Also, we let *def*$(B)$ be the index set of agents with a binding in B.

Given an SV-binding B, and a state, tree, or path formula ϕ, $B\phi$ is called a *bound formula*. $B\phi$ is a *bound literal* if ϕ is a path formula. A Boolean combination of bound literals is called a *Boolean bound formula* (*BB-formula*). The strategy variables in BB-formula are only used to tell whether two path properties are to be enforced with the same strategy. For example, property $\langle\{1\}\rangle((\langle+\{2\}\rangle\Diamond p)\wedge\langle+\{2\}\rangle\Diamond q)$ can be rewritten as BB-formula $(\{1\mapsto s_1,2\mapsto s_2\}\Diamond p)\wedge\{1\mapsto s_1,2\mapsto s_3\}\Diamond q$ which says that agent 1 must use the same strategy to fulfill both $\Diamond p$ and $\Diamond q$ while agent 2 may use different strategies to fulfill the two path properties.

Suppose that we are given a BB-formula ϕ, with strategy variables $s_1,\ldots,s_n$, and a function π that assigns each of $s_1,\ldots,s_n$ a strategy. Similar to the semantics of strategy logics [6] with strategy variables, we can also define the satisfaction of ϕ at a state q with π, in symbols $A,q\models^{\pi}\phi$, as follows.

Table 1. Rewriting rules for BB-formulas

$bf(B\neg\neg\phi)$	$\equiv bf(B\phi)$
$bf(\neg B(\phi_1 \vee \phi_2))$	$\equiv bf(B\neg\phi_1) \wedge bf(B\neg\phi_2)$
$bf(\neg B(\phi_1 \wedge \phi_2))$	$\equiv bf(B\neg\phi_1) \vee bf(B\neg\phi_2)$
$bf(B\neg\langle M\rangle\phi)$	$\equiv bf(B[M]\neg\phi)$
$bf(B\neg[M]\phi)$	$\equiv bf(B\langle M\rangle\neg\phi)$
$bf(B[M]\phi)$	$\equiv bf(B\langle[1,m]-M\rangle\phi)$
$bf(B\langle\{a_1,\ldots,a_n\}\rangle\phi)$	$\equiv bf(\{a_1 \mapsto newvar(),\ldots,a_n \mapsto newvar()\}\phi)$
$bf(B\neg\langle+M\rangle\phi)$	$\equiv bf(B[+M]\neg\phi)$
$bf(B\neg[+M]\phi)$	$\equiv bf(B\langle+M\rangle\neg\phi)$
$bf(B[+M]\phi)$	$\equiv bf(B^{\neg M} \circ \{a_1 \mapsto newvar(),\ldots,a_n \mapsto newvar()\}\phi)$, if $[1,m]-(def(B)\cup M)=\{a_1,\ldots,a_n\}$
$bf(B\langle+\{a_1,\ldots,a_n\}\rangle\phi)$	$\equiv bf(B \circ \{a_1 \mapsto newvar(),\ldots,a_n \mapsto newvar()\}\phi)$
$bf(B \bigcirc \phi)$	$\equiv B \bigcirc bf(\emptyset\phi)$
$bf(B\phi_1\mathrm{U}\phi_2)$	$\equiv Bbf(\emptyset\phi_1)\mathrm{U}bf(\emptyset\phi_2)$
$bf(B\phi_1\mathrm{W}\phi_2)$	$\equiv Bbf(\emptyset\phi_1)\mathrm{W}bf(\emptyset\phi_2)$
$bf(Bp) \equiv p$	; $bf(B\neg p) \equiv \neg p$
$bf(Btrue) \equiv true$	; $bf(Bfalse) \equiv false$

- $A,q \models^\pi \phi_1 \vee \phi_2$ iff either $A,q \models^\pi \phi_1$ or $A,q \models^\pi \phi_2$.
- $A,q \models^\pi \phi_1 \wedge \phi_2$ iff both $A,q \models^\pi \phi_1$ and $A,q \models^\pi \phi_2$.
- Given an SV-binding B and a path formula ϕ_1 with a C-strategy $\Sigma = \{a \mapsto \pi(s) \mid a \mapsto s \in B\}$, $A,q \models^\pi B\phi_1$ iff for all plays ρ compatible with Σ from q, $\rho \models_\Sigma \phi_1$.

In table 1, we present equivalence rules to rewrite state, tree, and path formulas to BB-formulas with procedure *bf*(). For convenience, we need a procedure *newvar*() that returns a strategy variable that has not been used before. The following lemma shows the correctness of the rules in table 1.

Lemma 10. *Given a state q and a BSIL formula ϕ with strategy variables $s_1,\ldots,s_n$ in $bf(\emptyset\phi)$, $A,q \models_\perp \phi$ iff there is a function π such that $A,q \models^\pi bf(\emptyset\phi)$.* ■

For convenience of algorithm presentation, 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 bound literals. The rewriting of a BB-formula ϕ to a DNBB-formula can be done with repetitive application of distribution law of conjunction to disjunction until no more change is possible.

Example 3. DNBB-formula rewriting: We have the following rewriting process for a BSIL formula for five agents.

$$\begin{aligned}
& bf\left(\emptyset\langle\{1\}\rangle\left(\begin{array}{c}\langle+\{2\}\rangle([+\{3\}](\langle+\{4\}\rangle\Box p) \vee [+\{2\}]\Diamond q)\\ \wedge\ [+\{3\}](\langle+2\rangle\Diamond r \wedge \langle+\{5\}\rangle\Box q)\end{array}\right)\right)\\
&\equiv bf\left(\{1 \mapsto s_1\}\left(\begin{array}{c}\langle+\{2\}\rangle([+\{3\}](\langle+\{4\}\rangle\Box p) \vee [+\{2\}]\Diamond q)\\ \wedge\ [+\{3\}](\langle+\{2\}\rangle\Diamond r \wedge \langle+\{5\}\rangle\Box q)\end{array}\right)\right)\\
&\equiv \left(\begin{array}{l}\ \ \ bf(\{1 \mapsto s_1\}\langle+\{2\}\rangle([+\{3\}](\langle+\{4\}\rangle\Box p) \vee [+\{2\}]\Diamond q))\\ \wedge\ bf(\{1 \mapsto s_1\}[+\{3\}](\langle+\{2\}\rangle\Diamond r \wedge \langle+\{5\}\rangle\Box q))\end{array}\right)\\
&\equiv \left(\begin{array}{l}\ \ \ bf(\{1 \mapsto s_1, 2 \mapsto s_2\}([+\{3\}](\langle+\{4\}\rangle\Box p) \vee [+\{2\}]\Diamond q))\\ \wedge\ bf(\{1 \mapsto s_1, 2 \mapsto s_3, 4 \mapsto s_4, 5 \mapsto s_5\}(\langle+\{2\}\rangle\Diamond r \wedge \langle+\{5\}\rangle\Box q))\end{array}\right)
\end{aligned}$$

$$\equiv \left(\begin{array}{l} \left(\begin{array}{l} \{1 \mapsto s_1, 2 \mapsto s_2, 4 \mapsto s_{13}, 5 \mapsto s_7\}\Box p \\ \vee \{1 \mapsto s_1, 3 \mapsto s_8, 4 \mapsto s_9, 5 \mapsto s_{10}\}\Diamond q \end{array}\right) \\ \wedge \{1 \mapsto s_1, 2 \mapsto s_{11}, 4 \mapsto s_4, 5 \mapsto s_5\}\Diamond r \\ \wedge \{1 \mapsto s_1, 2 \mapsto s_3, 4 \mapsto s_4, 5 \mapsto s_{12}\}\Box q \end{array}\right)$$

$$\equiv \left(\begin{array}{l} \left(\begin{array}{l} \{1 \mapsto s_1, 2 \mapsto s_2, 4 \mapsto s_{13}, 5 \mapsto s_7\}\Box p \\ \wedge \{1 \mapsto s_1, 2 \mapsto s_{11}, 4 \mapsto s_4, 5 \mapsto s_5\}\Diamond r \\ \wedge \{1 \mapsto s_1, 2 \mapsto s_3, 4 \mapsto s_4, 5 \mapsto s_{12}\}\Box q \end{array}\right) \\ \vee \left(\begin{array}{l} \{1 \mapsto s_1, 3 \mapsto s_8, 4 \mapsto s_9, 5 \mapsto s_{10}\}\Diamond q \\ \wedge \{1 \mapsto s_1, 2 \mapsto s_{11}, 4 \mapsto s_4, 5 \mapsto s_5\}\Diamond r \\ \wedge \{1 \mapsto s_1, 2 \mapsto s_3, 4 \mapsto s_4, 5 \mapsto s_{12}\}\Box q \end{array}\right) \end{array}\right) \text{; distribution of } \wedge \text{ over } \vee$$

This DNBB-formula sheds some light in analyzing BSIL formulas. As can be seen, the formula is satisfied iff one of the outermost disjuncts is satisfied. Without loss of generality, we examine the first disjunct: $\eta_1 \equiv \left(\begin{array}{l} \{1 \mapsto s_1, 2 \mapsto s_2, 4 \mapsto s_{13}, 5 \mapsto s_7\}\Box p \\ \wedge \{1 \mapsto s_1, 2 \mapsto s_{11}, 4 \mapsto s_4, 5 \mapsto s_5\}\Diamond r \\ \wedge \{1 \mapsto s_1, 2 \mapsto s_3, 4 \mapsto s_4, 5 \mapsto s_{12}\}\Box q \end{array}\right)$
There are the following three C-strategies involved in the satisfaction of the formula.

- Σ_1 for $\{1 \mapsto s_1, 2 \mapsto s_2, 4 \mapsto s_{13}, 5 \mapsto s_7\}$ of $\{1, 2, 4, 5\}$ used to satisfy $\Box p$.
- Σ_2 for $\{1 \mapsto s_1, 2 \mapsto s_{11}, 4 \mapsto s_4, 5 \mapsto s_5\}$ of $\{1, 2, 4, 5\}$ used to satisfy $\Diamond r$.
- Σ_3 for $\{1 \mapsto s_1, 2 \mapsto s_3, 4 \mapsto s_4, 5 \mapsto s_{12}\}$ of $\{1, 2, 4, 5\}$ used to satisfy $\Box q$.

This disjunct says the following interaction restrictions.

- Σ_1, Σ_2, and Σ_3 must agree in their choices at nodes owned by agent 1.
- Σ_2 and Σ_3 must agree in their choices at nodes owned by agent 4.

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 convenience of our algorithm presentation, we represent a conjunctive DNBB-formula η as a set of bound literals. Our goal is to design a computation tree exploration procedure that given a set Ψ of bound literals, labels each node in the tree with a subset of Ψ for the set of path formulas that some C-strategies have to enforce without violating the restrictions of strategy interaction imposed in Ψ through the strategy variables. In the design of the procedure, one central component is how to label the children of a node with appropriate subsets of Ψ as inherited path obligations. This is accomplished with the procedure *Suc_set*(A, q, Ψ) in the following. Given a node q in the computation tree and a set Ψ, the procedure nondeterministically returns an assignment of bound literals to children of q to enforce the bound literals in Ψ without violating the strategy interaction of bound literals.

```
Suc_set(A, q, Ψ) // λ() has been extended with satisfied child subformulas at each state.
1: Let $\Delta$ be $\{(q', \emptyset) \mid (q, q') \in R\}$ and $\Phi$ be $\Psi$.
2: **while** $\Phi$ is not empty **do**
3:     Pick an element $B\psi \in \Phi$ and let $\Phi$ be $\Phi - \{B\psi\}$.
4:     **if** ($\psi$ is either $\phi_1 U \phi_2$ or $\phi_1 W \phi_2$) with $\phi_2 \in \lambda(q)$ **then continue**.
```

```
 5:     else if ($\psi$ is either $\phi_1 \mathrm{U} \phi_2$ or $\phi_1 \mathrm{W} \phi_2$) with $\phi_1 \notin \lambda(q)$ then return $\emptyset$.
 6:     end if
 7:     if $\omega(q) \notin def(B)$ then
 8:        if $\psi$ is not $\bigcirc\phi_1$ then
 9:           for $(q', \Psi') \in \Delta$ do  replace $(q', \Psi')$ with $(q', \Psi' \cup \{B\psi\})$ in $\Delta$. end for
10:        else if there is an $(q, q') \in R$ with $\phi_1 \notin \lambda(q')$ then
11:           return $\emptyset$.
12:        end if
13:     else if $\exists B'\psi' \in \Psi - \Phi, \exists (q', \Psi') \in \Delta(B'\psi' \in \Psi' \wedge \omega(q) \in def(B) \cap def(B'))$
        then
14:        if $\psi$ is not $\bigcirc\phi_1$ then replace $(q', \Psi')$ with $(q', \Psi' \cup \{B\psi\})$ in $\Delta$;
15:        else if $\phi_1 \notin \lambda(q')$ then return $\emptyset$. end if
16:     else
17:        Nondeterministically pick a $(q', \Psi') \in \Delta$.
18:        if $\psi$ is not $\bigcirc\phi_1$ then replace $(q', \Psi')$ with $(q', \Psi' \cup \{B\psi\})$ in $\Delta$;
19:        else if $\phi_1 \notin \lambda(q')$ then return $\emptyset$. end if
20:     end if
21:  end while
22:  return $\Delta$.
```

The loop at statement 2 iterates through all the path obligations at the current node and passes them down to the children if necessary. Statements 4 checks if $B\psi$ is fulfilled. Statements 5 checks if $B\psi$ is violated. When a violation happens, the assignment of path obligations to children fails and we return $\emptyset$. The if-statement at line 7 is for nodes where no strategy choice is to be made. Statement 13 is for the case when we have already made a choice for a $B'\psi'$ that should share the same strategy decision with $B\psi$ at q. Thus the choice for $B\psi$ has to be consistent with that for $B'\psi'$. Statement 16 handles the case when there is no such $B'\psi'$.

5.2 Procedures for Checking BSIL Properties

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

```
Check_BSIL($A, q, \phi$)
1: if $\phi$ is $p$ then
2:    if $\phi \in \lambda(q)$ then  return true. else return false. end if
3: else if $\phi$ is $\phi_1 \vee \phi_2$ then
4:    return Check_BSIL($A, q, \phi_1$) $\vee$ Check_BSIL($A, q, \phi_2$)
5: else if $\phi$ is $\neg\phi_1$ then
6:    return $\neg$Check_BSIL($A, q, \phi_1$)
7: else if $\phi$ is $\langle M \rangle\theta$ for a tree or path formula $\theta$ then
8:    return Check_tree($A, q, \langle M \rangle\theta$)
9: end if
```

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

```
Check_set($A, Q', \phi_1$)
1: if $\phi_1 \notin P_A \cup \{true, false\}$ then
2:   for each $q' \in Q'$ do
3:     if Check_BSIL($A, q', \phi_1$) then
4:       Let $\lambda(q')$ be $(\lambda(q') \cup \{\phi_1\}) - \{\neg\phi_1\}$.
5:     else
6:       Let $\lambda(q')$ be $(\lambda(q') - \{\phi_1\}) \cup \{\neg\phi_1\}$.
7:     end if
8:   end for
9: end if
```

Then we use procedure Check_tree$(A, q, \langle M\rangle\theta)$ in the following to check if a state q satisfies $\langle M\rangle\psi$.

```
Check_tree($A, q, \langle M\rangle\theta$)
 1: Rewrite $bf(\emptyset\langle M\rangle\theta)$ to DNBB-formula $\eta_1 \vee \ldots \vee \eta_n$.
 2: for $i \in [1, n]$ do
 3:   Represent $\eta_i$ as a set $\Psi$ of bound literals.
 4:   for each $B\psi$ in $\Psi$. do
 5:     if $\psi$ is $\bigcirc\phi_1$ then
 6:       Check_set($A, \{q' \mid (q, q') \in R_A\}, \phi_1$).
 7:     else if $\psi$ is either $\phi_1 \mathrm{U} \phi_2$ or $\phi_1 \mathrm{W} \phi_2$ then
 8:       Check_set($A, Q_A, \phi_1$).
 9:       Check_set($A, Q_A, \phi_2$).
10:     end if
11:   end for
12:   if Rec_tree($A, q, \Psi$) then return true. end if
13: end for
14: return false.
```

We first rewrite $\langle M\rangle\theta$ to its DNBB-formula at statement 1 by calling $bf(\emptyset\langle M\rangle\theta)$ and using the distribution law of conjunctions over disjunctions. We then iteratively check with the loop starting from statement 2 if $\langle M\rangle\theta$ is satisfied due to one of its conjunctive DNBB-formula components of $\langle M\rangle\psi$. At statement 3, we construct the set Ψ of bound literals of the component. We evaluate the subformulas with the inner loop starting at statement 4. Finally at statement 12, we explore the computation tree, with procedure Rec_tree(A, q, Ψ) in the following, and pass down the path obligations to the children according to the restrictions of the SV-binding in Ψ.

```
Rec_tree($A, q, \Psi$)
1: if $(q, \Psi)$ coincides with an ancestor in the exploration then
2:     if there is no $B\phi_1 \mathrm{U} \phi_2$ in $\Psi$ then return true; else return false. end if
3: end if
4: Let Suc_set($A, q, \Psi$) be $\Delta$.
5: if $\Delta = \emptyset$ then return false end if
6: for each $(q', \Psi') \in \Delta$ with $\Psi' \neq \emptyset$ do
7:     if Rec_tree($A, q', \Psi'$) is false then return false. end if
8: end for
9: return true.
```

Note that procedure Rec_tree(A, q, Ψ) is nondeterministic since it employs *Suc_set*(A, q, Ψ) to nondeterministically calculate an assignment Δ of path obligations to the children of q.

5.3 Correctness Proof of the Algorithm

For the proof of the correctness of the algorithm, we define *strategy interaction trees* (*SI-trees*) in the following. An SI-tree for a set Ψ of bound literals and a game graph $A = \langle m, Q, I, \omega, P, \lambda, R \rangle$ from a state $q \in Q$ is a labeled computation tree $\langle V, r, \alpha, E, \beta \rangle$ with the following restrictions.

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

The SI-tree is *fulfilled* iff for every path $v_0 v_1 \ldots v_k \ldots$ along the tree from the root, there exists an $h \geq 0$ such that for every $j \geq h$, there is no $B\phi_1 \mathrm{U} \phi_2 \in \beta(v_j)$.

We have the following property between an SI-tree and execution of procedure Rec_tree(A, q, Ψ) from the root of an SI-tree.

Lemma 11. *For a set Ψ of bound literals,* Rec_tree(A, q, Ψ) *returns true iff there exists a fulfilled SI-tree for Ψ and A from q.* ■

Lemma 12. *Given a conjunctive DNBB-formula η represented as a set Ψ of bound literals, there exists a function π on strategy variables in η with $A, q \models^{\pi} \eta$ iff there exists a fulfilled SI-tree for A and Ψ from q.* ■

The correctness of procecure Rec_tree(A, q, Ψ) then directly follows from lemmas 11 and 12. Then with a structure induction on a given BSIL formula and the correctness of procedure Rec_tree(A, q, Ψ), we have the following lemma for the correctness of procedure Check_BSIL(A, q, ϕ).

Lemma 13. *Given a game graph A, a state q in A, and a BSIL formula ϕ,* `Check_BSIL`(A, q, ϕ) *iff* $A, q \models_{\perp} \phi$. ■

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 implement procedure `Rec_tree`(A, q, Ψ) with a stack of polynomial height. To see this, please be reminded that we use procedure *Suc_set*(A, q, Ψ) to calculate the assignment of bound literals to the children to q in the computation tree. Specifically, procedure *Suc_set*(A, q, Ψ) nondeterministically returns a set Δ with elements of the form (q', Ψ') such that $(q, q') \in R$ and $\Psi' \subseteq \Psi$. Thus along any path in the SI-tree, the sets of literal bounds never increase. Moreover, when there is a node in the exploration of SI-tree that coincides with an ancestor, we backtrack in the exploration. This implies that along any path segment longer than $|Q|$, either one of the following two conditions hold.

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

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

Lemma 14. *The BSIL model-checking algorithm in subsections 5.1 and 5.2 is in PSPACE.* ■

Following lemmas 9 and 14, we then get the following lemma.

Lemma 15. *The model-checking problem of a BSIL formula against a turn-based game graph is PSPACE-complete.* ■

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

6 Strategy Interaction Logic

In a BSIL formula, the topmost SIQ inside a path modal formula must not be a pure SQ. Lifting this restriction, we get *SIL* (*Strategy Interaction Logic*). For a game of m agents, an SIL formula ϕ has the following syntax.

$$\begin{aligned}\phi ::= p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \langle M\rangle\phi_1 \mid \langle M\rangle \bigcirc \phi_1 \mid \langle M\rangle\phi_1\mathrm{U}\phi_2 \mid \langle M\rangle\phi_1\mathrm{W}\phi_2 \\ \mid \langle +M\rangle\phi_1 \mid \langle +M\rangle \bigcirc \phi \mid \langle +M\rangle\phi_1\mathrm{U}\phi_2 \mid \langle +M\rangle\phi_1\mathrm{W}\phi_2\end{aligned}$$

The shorthands and semantics of SIL are exactly as those of BSIL. SIL is more expressive than ATL* for turn-based games. The reason is that SIQ "$\langle +\emptyset\rangle$" can be used to inherit strategies from parent modal operators to child ones. For example, $\langle\{1\}\rangle\Box\Diamond p$ is equivalent to $\langle\{1\}\rangle\Box\langle+\emptyset\rangle\Diamond p$.

Lemma 16. *SIL is strictly more expressive than ATL* for turn-based games.* ■

One implication of lemma 16 is that the model-checking problem of SIL is at least as hard as that of ATL*, which is doubly exponential time complete.

7 Conclusion

BSIL can be useful in describing close interaction 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.

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

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM (JACM) 49(5), 672–713 (2002)
2. Baier, C., Brázdil, T., Gröser, M., Kucera, A.: Stochastic game logic. In: QEST, pp. 227–236. IEEE Computer Society, Los Alamitos (2007)
3. Chatterjee, K., Henzinger, T.A., Piterman, N.: Strategy logic. Information and Computation 208, 677–693 (2010)
4. Costa, A.D., Laroussinie, F., Markey, N.: Atl with strategy contexts: Expressiveness and model checking. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010). Leibniz International Proceedings in Informatics (LIPIcs), vol. 8, pp. 120–132. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2010)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Company, New York (1979)
6. Mogavero, F., Murano, A., Vardi, M.Y.: Reasoning about strategies. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010). Leibniz International Proceedings in Informatics (LIPIcs), vol. 8, pp. 133–144. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2010)