An Extension of ATL with Strategy Interaction

FARN WANG, Graduate Institute of Electronic Engineering, National Taiwan University; Department of Electrical Engineering, National Taiwan University; Research Center for Information Technology Innovation (CITI), Academia Sinica, Taiwan, ROC
SVEN SCHEWE, Department of Computer Sciences, University of Liverpool
CHUNG-HAO HUANG, Graduate Institute of Electronic Engineering, National Taiwan University

We propose an extension to *ATL* (*alternating-time temporal logic*), called *BSIL* (*basic strategy-interaction logic*), for specifying collaboration among agents in a multiagent system. We show that BSIL is strictly more expressive than ATL$^+$ but incomparable with ATL*, *GL* (*game logic*), and *AMC* (*alternating μ-calculus*) in expressiveness. We show that a memoryful strategy is necessary for fulfilling a specification in BSIL. We establish that the BSIL model-checking problem is PSPACE-complete. However, BSIL model checking can be performed in time quadratic in the model for fixed formulas. The BSIL (and hence ATL$^+$) satisfiability is 2EXPTIME-complete. Finally, we report our experiment with a model checker for BSIL.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Games, concurrent, turn based, multiagent, strategy, logic, model checking, expressiveness, realizability, satisfiability

ACM Reference Format:
Farn Wang, Sven Schewe, and Chung-Hao Huang. 2015. An extension of ATL with strategy interaction. ACM Trans. Program. Lang. Syst. 37, 3, Article 9 (June 2015), 41 pages.
DOI: http://dx.doi.org/10.1145/2734117

1. INTRODUCTION

In the new era of web apps and mobile devices, the integrated development environment for various programming languages may face challenges for specifying and verifying computer systems that interact intensively with their users and their environment. The specification and verification of such open systems focus on the design of system interfaces that allow for the fulfillment of some objectives of the users while enforcing certain safety policies of the systems. Such open systems can naturally be modeled as games, and their analysis therefore benefits from game-based techniques. From a game theoretical perspective, the design problem of such open systems can be modeled as multiagent games. Some agents represent the system, while other agents represent the environment and the users of the system. An agent may have several objectives,

This work is supported by grant MOST 103-2221-E-002-150-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.
Authors' addresses: F. Wang, Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan 106, ROC; email: farn@ntu.edu.tw; S. Schewe, Department of Computer Sciences, University of Liverpool, Liverpool, UK; email: Sven.Schewe@liverpool.ac.uk; C.-H. Huang, Graduate Institute of Electronic Engineering, National Taiwan University, Taipei, Taiwan 106, ROC; email: yyergg@gmail.com.

DOI: http://dx.doi.org/10.1145/2734117

and these objectives can conflict with the objectives of other agents. On the one hand, if a user objective is not in conflict with the system's objectives, then the user should be allowed to achieve this 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 his or her bank account, check balances of his or her account, and so forth. The user may also have other objectives forbidden by the system, such as withdrawing money from the bank accounts of other customers or checking their balances. A system is then well designed—reflected by winning in an execution (or *play* in the jargon of game theory)—if the user can meet his or her allowed objectives but he or she cannot meet his or her disallowed objectives. The goal in system design is, from the game theoretical perspective, to design a computable strategy of the system that enforces all the system objectives, allows the user to achieve objectives that are consistent with the system objectives, and prevents the user from achieving objectives that are incompatible with the system objectives. Such a strategy is called a *winning strategy* for the system.

At the moment, there are various logics that express such properties of strategic power of agents, including *ATL* (*alternating-time logic*), *ATL**, *AMC* (*alternating* μ-*calculus*), *GL* (*game logic*) [Alur et al. 2002], ATL$_{sc}$, ATL$^*_{sc}$ [Costa et al. 2010], and *SL* (*strategy logics*) [Chatterjee et al. 2010; Mogavero et al. 2010], 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. In particular, 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 [Alur et al. 2002] allow us to specify that some players together have a strategy to satisfy some fully temporalized objective: strategy quantifiers mark the start of a state formula. As exemplified later, this is far from what the industry needs in specification.

Consider the example of a bank that would like to specify its information system embodied as a system security strategy that 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 his or her 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 that are considered "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* [Alur et al. 2002] 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 previous example, strategy logics have been proposed in Costa et al. [2010], Chatterjee et al. [2010], and Mogavero et al. [2010]. These strategy logics allow for the flexible quantification of high-order strategy variables in logic formulas. However, their verification complexities are prohibitively high and hinder their practical application. The high complexity is due to the freedom in the use of these strategies: the strategies in such strategy logics can be combined in unrestricted ways. For example, in the strategy logic introduced in Mogavero et al. [2010], we can write down the following 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 Z, respectively. $[\![Y]\!]$ is a universal quantification on a strategy named Y. The operators $(1, X)$, $(2, Y)$, $(2, Z)$, and $(3, Z)$ bind strategy X, Y, Z, and Z, to Agents $1, 2, 2$, and 3, respectively. It is apparent that this language provides a very high level of freedom in reasoning over strategies. As frequently witnessed in the history of temporal logic development, proper restrictions in the modal operations and path quantifiers can lead to a spectrum of sublogics with different expressiveness and model-checking efficiency. The following are two examples:

- The validity problem of $\mathcal{L}$ (the first-order language of unary predicate symbols and the binary predicate symbol $\leq$) is nonelementary [Stockmeyer 1974], while PTL, with the same expressiveness as $\mathcal{L}$, is only PSPACE-complete [Sistla and Clarke 1985].
- Between CTL [Clarke et al. 1986] and CTL* [Emerson and Halpern 1985, 1986], there are many subclasses of CTL* with various balances between expressiveness and verification efficiency. Fair CTL [Emerson and Lei 1987], 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 [Ben-Ari et al. 1983; Emerson and Clarke 1980; Emerson and Halpern 1986; Lamport 1980].

Subclasses of temporal logics with proper balance between expressiveness and verification efficiency have thus proven to be of practical relevance in addition to being theoretically interesting. Indeed, most specifications in real-world projects have simple structures, for example, safety, liveness, and so forth. It is therefore interesting to identify and study "natural" subclasses of strategy logics with a proper balance between expressiveness and model-checking efficiency. Moreover, it appears to be practical and appealing if the identified subclass can be characterized with elegant and intuitive syntax. It is the main purpose of this article to propose new natural modal operators for strategy collaborations and extend ATL for a subclass of strategy logic, which provides a good balance between expressiveness and efficiency.

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

Example 1.1. **Prisoners' dilemma.** Suppose the police are interrogating three suspects (prisoners). The police have very little evidence. A prisoner may cooperate (with his or 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 or 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 he or she will be a "dirty witness" and all charges against him or 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 et al.'s [2002] ATL*, GL, or AMC, respectively, as follows:[1]

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

[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. Operator "**lfp**" is the least fix-point 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 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 prisoners' 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. (See [Ben-Ari et al. 1983; Emerson and Clarke 1980] 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 prisoners' dilemma to explain BSIL, starting with the following specification for the property discussed at the end of Example 1.1:

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

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 "$\langle +\rangle$" in the following.) 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 that seems natural for practical specification. Consider another example. If Prisoner 1 really hates the others, he or she can always betray the other prisoners, making sure that Prisoners 2 and 3 will be in jail, and let them decide whether he or she will be in jail too. This property can be expressed in BSIL as follows:

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

For convenience of discussion, we introduce the term *strategy profile*, which is technically a partial function from agent indices to strategies. A strategy profile represents the collaboration among defined agents in the profile to fulfill a specification. Formula (B) exhibits how BSIL can be useful in specifying the combination and interaction of strategy profiles for various goals. Specifically, the following three strategy profiles are used in Formula (B):

- The first, say, Σ, is defined only on 1 and fulfills $\Box(j_2 \wedge j_3)$.
- The second, say, Π_1, is defined on 1, 2, 3 and fulfills $\Diamond\neg j_1$.
- The second, say, Π_2, is defined on 1, 2, 3 and fulfills $\Box j_1$.

Moreover, Σ, Π_1, and Π_2 use the same strategy for Agent 1.

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. In fact, BSIL admits one alternation of strategy quantification. Thus,

we can use BSIL for the specification of the ways to combine 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 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 SIQ may have to be computed dynamically. Thus, the explicit universal SIQs will not only necessarily blow up the verification complexity but also contradict the mainstream of game theory for statically calculable strategies. In contrast, some strategy logics [Mogavero et al. 2010] 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 [Chatterjee et al. 2010; Costa et al. 2010; Mogavero et al. 2010] are superclasses to BSIL with their flexible quantification of strategies and binding to strategy variables, their model checking[2] complexities are all 2EXPTIME-hard. In contrast, BSIL enjoys a PSPACE-complete model-checking complexity for turn-based and concurrent game graphs. This may imply that BSIL provides a better balance between expressiveness and verification efficiency than ATL*, GL, AMC [Alur et al. 2002], and SL [Chatterjee et al. 2010; Mogavero et al. 2010]. Further related work is the stochastic game logic (SGL) by Baier et al. [2007], which allows for expressing strategy interaction. However, for memoryful strategies, the model-checking problem of SGL is undecidable.

We also establish some additional 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 needs to 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 realizability problem is complete for doubly exponential time.

The article is organized as follows. Section 2 discusses related work. Section 3 uses the banking system as a running example. Section 4 explains concurrent and turn-based game graphs for the description of multiagent systems. Section 5 presents BSIL and ATL$^+$ and establishes the need for memoryful strategies. Section 6 discusses the expressiveness of BSIL. Section 7 establishes the PSPACE-completeness of the model-checking problem for BSIL. In Section 8, we present the alternative model-checking algorithm based on weak alternating automata construction. Section 9 discusses the complexity of the BSIL realizability problem. Section 10 reports our implementation and experiments. Section 11 is the conclusion.

2. RELATED WORK

2.1. Prior to Strategy Logics

Kupferman et al. [2001] proposed module checking, a famous framework for checking whether open systems satisfy temporal logic properties. In this framework, the open system interacts with its environment. For LTL, an open system satisfies a property if there is no execution of the system that violates the specification. The two views of (1) unraveling the system to the tree of its executions and checking if it contains an infinite path that violates the specification and (2) checking if the environment has a strategy to inflict a violation seem to be interchangeable. For branching time logics, however, this is different in model checking.

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

Checking if the unraveling of the system satisfies a specification is the traditional target of model checking. In a nutshell, the critique raised in Kupferman et al. [2001] is that checking that this tree is a model does not guarantee that it works in every environment. It is argued that, for this, every subtree (without dead ends) of this tree should satisfy the specification, as an environment may not be able to display all the actions allowed for by the model. Testing whether a module comes with such strong guarantees is referred to as module checking. Module checking can thus be seen as bringing environment strategies to model-checking branching time logics, as the question of whether or not there is a pruned tree that is not a model of ϕ relates to the question of whether or not there is a pruning strategy for the environment that produces a tree that is not a model of ϕ.

For branching time logics, the existence of such a nondeterministic environment strategy is a solvability problem, and the construction of a witness strategy is a synthesis problem. It is therefore not surprising that the usual exponential blow-up is incurred. The related synthesis problems in module checking is whether there is an open system that provides these strong guarantees. They are referred to as synthesis in reactive environments [Kupferman et al. 2000; Schewe and Finkbeiner 2007] since the underlying concept is the same as those in the newer game logics from the ATL family. That is, for path properties like LTL specifications, we are still interested at calculating a strategy for the system so that, for all counter strategies of the environment, the path properties are maintained. In contrast, the pruning strategy for branching time logics play a different role in that they shape the tree. For branching time logics, we could view this as a two-player game, where the strategy of the "environment player" to shape the tree is a strategy that ranges over the other strategy quantifiers (in the form of existential and universal path quantifiers).

Alur et al. [2002] presented ATL (alternating-time temporal logic), ATL*, AMC (alternating-time μ-calculus), and GL (game logic) with strategy quantifier $\langle\langle M \rangle\rangle$.

Brihaye et al. [2009] introduced a very expressive extension to ATL* with other players' strategies and memory constraints. They also showed that the model-checking problem of this extension is decidable.

Pinchinat [2007] introduced a way to specify expressive constraints on strategies in concurrent games by extending μ-calculus with decision modalities. Laroussinie and Markey [2013] reported decidability of satisfiability problems in this direction with new context constraints (e.g., the number of moves by agents is bound).

2.2. With Strategy Logics

Chatterjee et al. [2010] introduced a strategy logic allowing for first-order quantification over strategies. The decision procedure is, however, nonelementary.

Mogavero et al. also identified fragments of strategy logics that enjoy a doubly exponential time complexity model-checking algorithm [Mogavero et al. 2010, 2012, 2013]. For example, in Mogavero et al. [2013], conjunction and disjunction cannot happen in the same scope of strategy quantification. BSIL is also a fragment of strategy logic with restriction on the hierarchy of SIQs and sometimes can be handy in expressing the Boolean relation among strategies. Moreover, the restriction on BSIL results in a much lower model-checking complexity of PSPACE.

Another interesting extension of ATL are the logics ATL$_{sc}$ and ATL$^*_{sc}$, introduced by Costa et al. [2010], which interprets nested strategy quantification as a composition—rather than a revocation—of strategy profiles. This is similar to the semantics of our SIQs, and BSIL is indeed a sublogic of ATL$^*_{sc}$. The difference in the design of these logics is the goal. The increased freedom in binding and freeing strategies in ATL$_{sc}$ leads to a nonelementary model-checking procedure [Costa et al. 2010].

The flexibility in nesting strategy quantification also leads to equal expressiveness of ATL$_{sc}$ and ATL$^*_{sc}$ [Costa et al. 2010], whereas BSIL cannot express ATL*. Naturally,

this freedom in ATL_{sc} comes at a cost, namely, in the form of the high complexity of the decision problems of these logics. In contrast, BSIL has been designed with expressiveness and complexity in mind, in particular by restricting the use of negations and by allowing for binding, but not for revoking the binding of strategies by SIQs. These restrictions keep the model-checking complexity at bay (PSPACE) while maintaining the capability to express relevant properties.

BSIL and the various fragments of strategy logics all can express certain interaction and collaboration relations among strategies used by agents in a multiagent game. They are designed with different considerations in balancing between expressiveness and verification efficiency. BSIL is also special in that its design takes the natural way that strategy collaboration can be specified into consideration.

3. RUNNING EXAMPLE

We use the banking example described in the introduction to explain the idea of this work. Suppose that a bank wants to provide the following services:

- Depositing to an account by a client from the root screen
- Transferring money from an account in the bank to an account in another bank, also from the root screen

As we can see, the deposit service involves the interaction between a client and the bank, while the transfer service involves at least three parties: a client, the bank, and a partner bank. Also, in the meantime, the bank wants to forbid any client and partner from checking sensitive information of other clients, for example, checking the password of another client.

3.1. Trying to Write Down a Correct Formal Specification

To develop the services, the bank manager in charge of the project needs to specify the services and make sure that the specification is not erroneous. If he or she turns to the literature, the bank manager may choose ATL or its extensions, such as fair ATL, ATL*, AMC, or GL, to specify the services. The choice is plausible at first glance since ATL and fair ATL both have a polynomial-time model-checking algorithm and could support the verification of the services. So the manager could attempt to specify the services with the following formula:

$$\langle 1\rangle \begin{pmatrix} \Box\neg \mathit{checkOthersPassword} \\ \wedge\ \langle 2\rangle\Diamond \mathit{depositDone} \\ \wedge\ \langle 2,3\rangle\Diamond \mathit{transferDone} \end{pmatrix}. \tag{C}$$

Here, we use the agent with index/name 1 for the bank, 2 for the client, and 3 for the partner bank. But on second glance, we can see that this specification is too permissive. The subformula $\langle 2\rangle\Diamond \mathit{depositDone}$ says that the client can force a deposit transaction no matter how the banking system responds. In practice, there are many factors beyond the control of the client for the success of a transaction. For example, when the banking system is in maintenance or out of order, the transaction may fail.

After realizing the problem with Formula (C), the manager may decide to rewrite the specification as follows:

$$\begin{array}{ll} & \langle 1\rangle\Box\neg \mathit{checkOthersPassword} \\ \wedge & \langle 1,2\rangle\Diamond \mathit{depositDone} \\ \wedge & \langle 1,2,3\rangle\Diamond \mathit{transferDone} \end{array}. \tag{D}$$

This specification repairs the issue observed in Formula (C) since now $\langle 1,2\rangle\Diamond \mathit{depositDone}$ says with the collaboration of the banking system, the client can finish a deposit transaction. However, there is a subtle issue that nullifies the formula.

In the collaboration between the banking system and the client for property $\langle 1,2\rangle\Diamond \mathit{depositDone}$, the banking system is no longer required to maintain property $\Box\neg \mathit{checkOthersPassword}$ since, according to the semantics of ATL, the specification does not require the banking system to use the same strategy to enforce both $\langle 1\rangle\Box\neg \mathit{checkOthersPassword}$ and $\langle 1,2\rangle\Diamond \mathit{depositDone}$. The same issue also appears in $\langle 1,2,3\rangle\Diamond \mathit{transferDone}$. That is, the banking system's strategy in enforcing $\langle 1,2,3\rangle\Diamond \mathit{transferDone}$ is not required to be the same one as in enforcing $\langle 1\rangle\Box\neg \mathit{checkOthersPassword}$. Thus, the service system would be allowed to leak the client's passwords when transferring funds.

3.2. Resorting to General Strategy Logics for a Correct Specification

When the manager discovers after several trials that the specification cannot be expressed in ATL, ATL*, GL, and AMC [Alur et al. 2002], he or she may turn to strategy logics [Chatterjee et al. 2010; Mogavero et al. 2010] from the literature. Using these logics, he or she can express the property as follows:

$$\langle x\rangle\langle y\rangle\langle z\rangle\langle w\rangle[v][u][t]\begin{pmatrix}(1,x)(2,v)(3,u)\Box\neg \mathit{checkOthersPassword}\\ \wedge\ (1,x)(2,y)(3,t)\Diamond \mathit{depositDone}\\ \wedge\ (1,x)(2,z)(3,w)\Diamond \mathit{transferDone}\end{pmatrix}. \tag{E}$$

This formula declares four existentially quantified strategies, x, y, z, and w, and three universally quantified strategies, v, u, and t, and says the following:

- On using strategy x, the banking system can enforce $\Box\neg \mathit{checkOthersPassword}$.
- On using strategy x and y, respectively, the banking system and a client can finish the deposit transaction.
- On using strategy x, z, and w, respectively, the banking system, the client, and a partner bank can also finish a fund-transfer transaction.

However, when the manager is pleased with this elegant specification, he or she may want to use it to verify that the specification is met by the system after development or delivery. In this case, he or she would find that there is no tractable algorithm and no working tools for checking general specifications in strategy logics or synthesizing the strategies. In fact, the complexity reported in the literature is at best doubly exponential [Chatterjee et al. 2010; Mogavero et al. 2010]. So the manager finds him- or herself in the dilemma between expressiveness and verification efficiency.

A further theoretical problem that has been discussed in the literature is that strategy logics offer a reasoning power that may encode unexpected behavior. Let us consider the following example:

$$\langle x\rangle\langle y\rangle[z]\langle w\rangle[v][u][t]\begin{pmatrix}(1,x)(2,v)(3,u)\Box\neg \mathit{checkOthersPassword}\\ \wedge\ (1,x)(2,v)(3,t)\Box(2,y)\Diamond \mathit{depositDone}\\ \wedge\ (1,x)(2,u)(3,t)\Box(2,z)(3,w)\Diamond \mathit{transferDone}\end{pmatrix}. \tag{F}$$

Here, $[z]$, $[v]$, $[u]$, and $[t]$ universally quantify strategies named z, v, u, and t, respectively. As observed by Mogavero et al. [2014], these strategies include nonbehavioral strategies: a choice of an agent, at a given point of a play, may depend on choices other agents can make in the future or in counterfactual plays. As the latter moves are unpredictable, such strategies cannot easily be implemented, making the use of the logic problematic in practice. Indeed, the huge verification complexity of strategy logics can be attributed to such freestyle binding operations of strategy names to agents. Thus, the manager could wish for a subclass of strategy logics that allows for elegant expressions of natural and practical specifications while supporting verification with less complexity. Similar examples can, in fact, appear in different projects with services that rely on the collaboration of multiple agents.

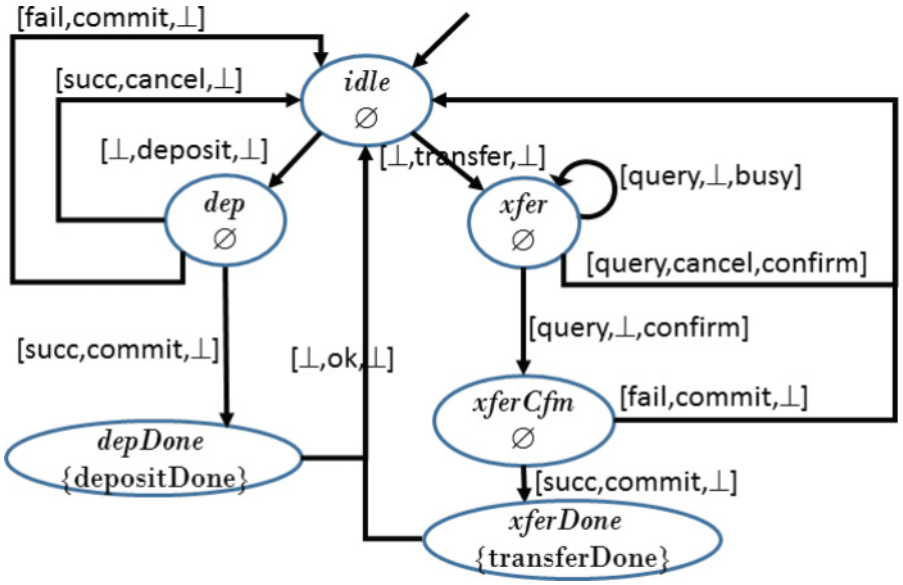


Fig. 1. Game graph of a banking system.

3.3. BSIL: New Strategy Modalities Expressive Enough for the Specification

In fact, in conceivable applications, the developers cannot implement and are not interested in clairvoyant strategies. In other words, for practical specifications, it is likely that existential SIQs are expressive enough. In BSIL, Formula (E) can be rewritten as follows:

$$\langle 1\rangle \begin{pmatrix} \Box\neg \mathit{checkOthersPassword} \\ \wedge\ \langle +2\rangle \Diamond \mathit{depositDone} \\ \wedge\ \langle +2, 3\rangle \Diamond \mathit{transferDone} \end{pmatrix}. \tag{G}$$

This formula says that the banking system has a strategy in which at any instant, the system can ensure that no password is leaked, the system allows a client to finish a deposit transaction, and the system allows a client to transfer funds from or to a partner bank.

3.4. Symbolic Strategy Names and Path Obligations

But to verify such a BSIL property, we need new techniques to check that, along plays, the same decision is made for the same strategy. Consider the model of the banking system in Figure 1. We need to find strategies of the agents to fulfill $\Box\neg \mathit{checkOthersPassword}$, $\Diamond \mathit{depositDone}$, and $\Diamond \mathit{transferDone}$. To explain our verification algorithms, we use symbolic strategy names x, y, z, w and the binding notations in Formula (G). In fact, Formula (G) is exactly the same as Formula (E). Thus, by interpreting all symbolic strategy names as existentially quantified, Formula (G) can be rewritten as follows:

$$\begin{aligned} &((1, x)\Box\neg \mathit{checkOthersPassword}) \\ \wedge\ &((1, x)(2, y)\Diamond \mathit{depositDone}) \\ \wedge\ &((1, x)(2, z)(3, w)\Diamond \mathit{transferDone}). \end{aligned} \tag{H}$$

Note that, since we only allow explicit existential strategy quantification, the number of strategy bindings are exactly determined by the number and sizes of the SQs and SIQs in the formula.

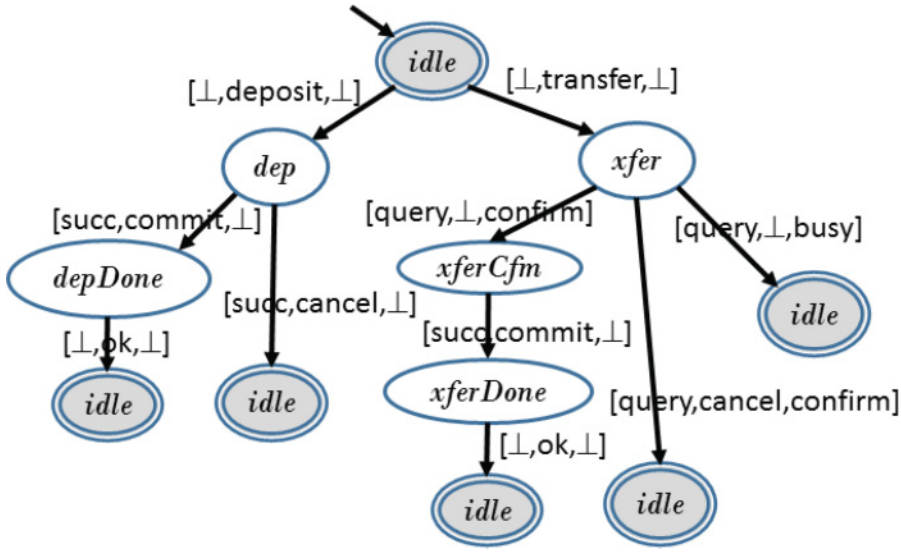


Fig. 2. Part of the computation tree of the banking system.

If we label $(1,x)\Box\neg checkOthersPassword$, $(1,x)(2,y)\Diamond depositDone$, and $(1,x)(2,z)(3,w)\Diamond transferDone$ on nodes of a computation tree in a labeling procedure, the symbolic names precisely reflect the constraints on the decisions along plays. For example, suppose that both $(1,x)\Box\neg checkOthersPassword$ and $(1,x)(2,y)\Diamond depositDone$ are labeled on a state of location *dep*. This state has three successor states: for convenience C1 of location *idle* via [fail, commit, $\perp$], C2 of location *idle* via [succ, cancel, $\perp$], and C3 of location *depDone* via [succ, commit, $\perp$]. If strategy x chooses action succ, then obligation $(1,x)\Box\neg checkOthersPassword$ will be passed down to C2 and C3. In this situation, whether obligation $(1,x)(2,y)\Diamond depositDone$ is labeled on C2 or C3 relies on the action decision of strategy y. But the constraint on the strategy decision is that if $(1,x)(2,y)\Diamond depositDone$ is labeled on C2, then $(1,x)\Box\neg checkOthersPassword$ must also be labeled on C2, and if $(1,x)(2,y)\Diamond depositDone$ is labeled on C3, then $(1,x)\Box\neg checkOthersPassword$ must also be labeled on C3. The reason is that the path obligations are enforced by the same strategy of the bank.

3.5. Passing Down the Path Obligations While Observing the Restrictions Among S-Profiles

The previous observation points out how to judge whether a passing-down scheme of path obligations from a parent state to its child states is consistent with the strategy quantification. For example, suppose that we have the following characteristics of the strategy profile $(1,x)(2,y)(2,z)(3,w)$:

- Strategy x of the banking system never issues action fail.
- Strategies y and z never cancel transactions for a client.
- Strategy w never shows a busy message for the partner bank.

Then, the verification problem of strategy interaction can be visualized as finding strategies x, y, z, and w that select paths in a computation tree to satisfy three path obligations: $\Box\neg checkOthersPassword$, $\Diamond depositDone$, and $\Diamond transferDone$. Figure 2 shows part of a computation tree when strategy x for the banking system is fixed with the following restriction:

- At location *idle*, *depDone*, and *xferDone*, always issue action $\perp$.
- At location *dep* and *xferCfm*, only issue action succ.

The leaves of the tree all end at location *idle*, where the same tree structure is replicated. We can start synthesizing the strategies y, z, and w in the context of this strategy x, which, for convenience of explanation, is memoryless (or positional).

As we have suggested, we can label the path obligations on the nodes of the computation tree and examine whether a strategy profile can fulfill all the path obligations. In the literature, we can name the nodes by their branching paths. For example, the root is named ε, which is the empty string. Then, the node at location *xferDone* is named "100" and the rightmost *idle* node is named "12." We begin by label node ε with the three path obligations: $(1, x)\Box\neg$*checkOthersPassword*, $(1, x)(2, y)\Diamond$*depositDone*, and $(1, x)(2, z)(3, w)\Diamond$*transferDone*. Then we can pass the path obligations to the children in the following way:

- We must pass $(1, x)\Box\neg$*checkOthersPassword* to both child "0" and "1" of the root since the two children are both selected by action $\perp$ of the banking system.
- We can pass $(1, x)(2, y)\Diamond$*depositDone* and $(1, x)(2, z)(3, w)\Diamond$*transferDone*, respectively, to child "0" and "1" of the root since the two children are selected with the same strategy x of the banking system and different strategies (y and z) of the client.

Note that we can check whether the passing down of the path obligation is consistent with the strategy quantifications in the input Formula (G) just by checking the labeling of path obligations with strategy name bindings since all consistency information is maintained there.

Similarly, from node "0," strategy x and y can together choose to pass $(1, x)(2, y)\Diamond$*depositDone* to node "00," while strategy x must unilaterally pass $(1, x)\Box\neg$*checkOthersPassword* to both "00" and "01." Then, at node "00," $(1, x)(2, y)\Diamond$*depositDone* is fulfilled eventually and no longer needs be passed down. The passing down of path obligations from node "1" and fulfillment of $(1, x)(2, z)(3, w)\Diamond$*transferDone* can then be shown similarly.

3.6. Finding Finite Satisfying Evidence for a Formula in a Computation Tree

As we can see, the existence of the tree top in Figure 2 is sufficient for synthesizing strategy profile $(1, x)(2, y)(2, z)(3, w)$ to satisfy Formula (G). From this example, it is easy to see the requirement for such a tree top. First, all eventual-formulas (or until-formulas) from the root SQ to the next level of SQs have to be fulfilled by the strategy profile. Second, the leaves of the tree top do not contain any eventual-formula (or until-formula) labels. Thus, our PSPACE algorithm for model checking BSIL formulas is actually a nondeterministic one that guesses and checks the consistency of the tree top and strategic actions at the nodes in the tree top.

Then, the final question regarding our algorithms is the terminating condition for searching the tree tops and strategy profiles. Let us consider a tree node labeled with path obligations and how to (nondeterministically) explore for a tree top and a strategy profile. Note that the path obligations labeled on a child will be no more than those of its parent. In fact, the path obligations can stop being passed down in a path when they are fulfilled. Path obligations are simply passed down and not generated along the paths. As a result, along a path longer than the size of the game graph, either the number of path obligations decreases or two nodes, say, v and v' (for convenience, we assume v is an ancestor of v'), with identical location and labels, must appear. If the tree top is evidence of the existence of a strategy profile and the latter happens, then we can replace the subtree rooted at v with the one rooted at v' and the resulting tree top is still evidence. This observation implies that we can focus on tree tops such that along any path, there is no duplication of nodes with the same location and the same

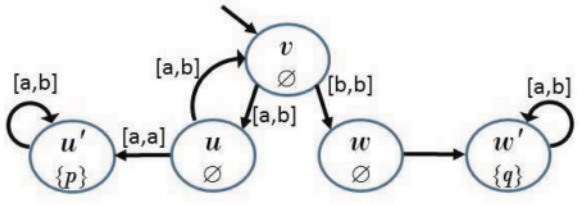


Fig. 3. A concurrent game graph.

path obligation labels. This implies that we only have to explore tree tops of depth no greater than the size of the game graph times the number of temporal modalities in the given BSIL formula.

4. GAME GRAPHS

4.1. Concurrent Games

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

Definition 4.1. A concurrent game graph (CGG) is a tuple $A = \langle m, Q, r, P, \lambda, R, \Delta, \delta \rangle$ with the following restrictions:

- 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 3, we have the graphical representation of a concurrent game graph. The ovals represent states, while 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 3, on edge (v, u), we have label $[a, b]$, which means that to make the transition, Agent 1 has to choose token a while Agent 2 has to choose b.

For convenience, in the remainder of the article, 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 game graph $\mathcal{G}$.

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 $last(\rho)$ to denote the last state in ρ, that is, $\rho(|\rho| - 1)$.

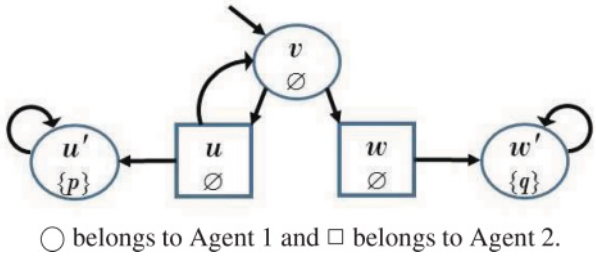


Fig. 4. A turn-based game graph.

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 $a \in [1, m]$, $a \in A$ if and only if $\Sigma(a)$ 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 the 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 $a \in [1, m]$ if and only if 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 if and only if for every $a \in A$, the play is compatible with $\Sigma(a)$ of Agent a.

4.2. Turn-Based Games

Another popular game structure is the *turn-based game,* in which at each state, at most one agent gets to decide the next state. For example, in Figure 4, 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.

In fact, turn-based games are special cases of concurrent games, as every turn-based game can be represented as a concurrent game. Specifically, a turn-based game graph (TBG) $\mathcal{G} = \langle m, Q, r, P, \lambda, R, \Delta, \delta \rangle$ can be viewed as a concurrent game, which is defined as follows:

- $\Delta = Q \cup \{\perp\}$, 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 $(q_1, q_2), (q_1, q_3) \in R$, and Agent a, $\delta((q_1, q_2), a) = \perp$ if and only if $\delta((q_1, q_3), a) = \perp$. 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.

Turn-based game graphs are often easier to handle than concurrent games. For this reason, we will often use turn-based game graphs in examples and explanation of the theory in order to ease the presentation.

5. BASIC STRATEGY INTERACTION LOGIC (BSIL)

5.1. 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* and $\langle +A\rangle$ is a *strategy interaction quantifier*. $\langle A\rangle\psi$ 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. Also, for conciseness, we omit null SIQs $\langle +\rangle$.

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:

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

The SQs and SIQs introduced earlier 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 notation, 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.

5.2. 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:

- $\mathcal{G}, q \models_\Sigma p$ if and only if $p \in \lambda(q)$.
- For state formula ϕ_1, $\mathcal{G}, q \models_\Sigma \neg\phi_1$ if and only if $\mathcal{G}, q \models_\Sigma \phi_1$ is false.
- For state or tree formulas ψ_1 and ψ_2, $\mathcal{G}, q \models_\Sigma \psi_1 \wedge \psi_2$ if and only if $\mathcal{G}, q \models_\Sigma \psi_1$ and $\mathcal{G}, q \models_\Sigma \psi_2$.
- For state or tree formulas ψ_1 and ψ_2, $\mathcal{G}, q \models_\Sigma \psi_1 \vee \psi_2$ if and only if either $\mathcal{G}, q \models_\Sigma \psi_1$ or $\mathcal{G}, q \models_\Sigma \psi_2$.
- $\mathcal{G}, q \models_\Sigma \langle A\rangle\tau$ if and only if there exists an S-profile Π of A with $\mathcal{G}, q \models_\Pi \tau$.
- $\mathcal{G}, q \models_\Sigma \langle +A\rangle\tau$ if and only if 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.

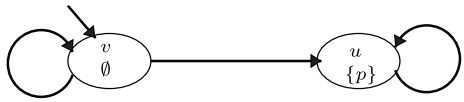


Fig. 5. A simple turn-based game that demands memoryful strategies.

- $\mathcal{G}, q \models_\Sigma \langle A\rangle\theta$ if and only if 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.
- $\mathcal{G}, q \models_\Sigma \langle +A\rangle\theta$ if and only if 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 $\rho \models_\Sigma \theta$, if and only if the following restrictions hold:

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

For convenience, we let $\perp$ be a null S-profile, that is, 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 $\mathcal{G} \models \phi_1$.

5.3. ATL$^+$

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

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

ATL$^+$ can also be viewed as an extension of ATL [Alur et al. 2002] that, similar to the extension of CTL to CTL$^+$ [Ben-Ari et al. 1983; Emerson and Halpern 1986], allows for the Boolean combination of path formulas. 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.

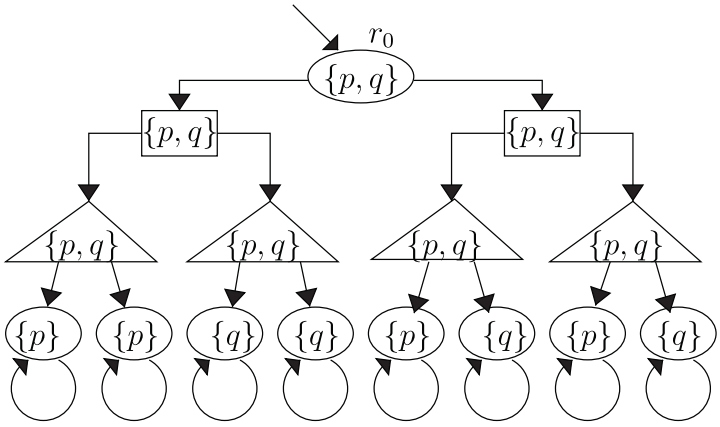
5.4. Memory

In this subsection, we show a simple example, in which the agents need memory to achieve their objective for ATL$^+$ specifications. This is exemplified by the simplest possible case: the turn-based game in Figure 5 with one agent, two states, one atomic proposition, and two memoryless strategies that do not count the unreached states in the histories. For the ATL$^+$ sentence $\langle 1\rangle((\neg\bigcirc p) \wedge \Diamond p)$, apparently Agent 1 needs memory to enforce it. In fact, we can propose another semantics of ATL$^+$ (and thus BSIL) that only allows memoryless strategies in all strategy profiles. The earlier example can be used to establish the following lemma.

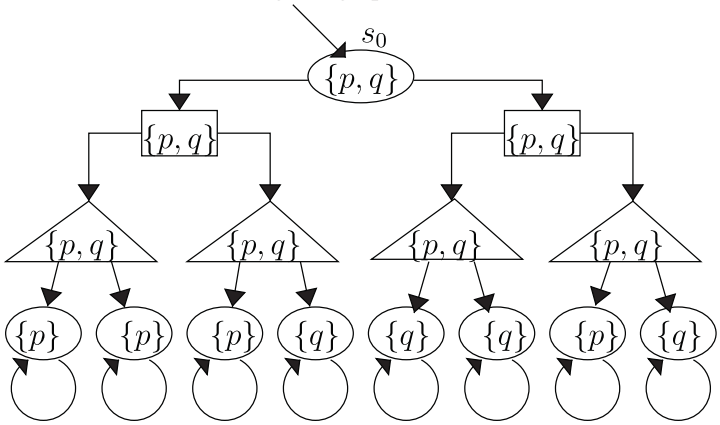
LEMMA 5.1. *The semantics of ATL$^+$ (and thus BSIL) with memoryless strategies is not equivalent to the original semantics. This even holds for the single-agent case.*

6. EXPRESSIVE POWER OF BSIL

In this section, we establish that BSIL is incomparable with ATL*, AMC, and GL [Alur et al. 2002] in expressiveness. In fact, we will first establish the incomparability between BSIL with GL and AMC. Then, the incomparability with BSIL follows since GL and AMC are superclasses of ATL* [Alur et al. 2002].



(a) G_1, a game graph for base case.



(b) H_1, another game graph for base case.
◯ belongs to Agent 1; □ belongs to Agent 2; and △ belongs to Agent 3.

Fig. 6. Base cases for the expressiveness of BSIL over ATL*.

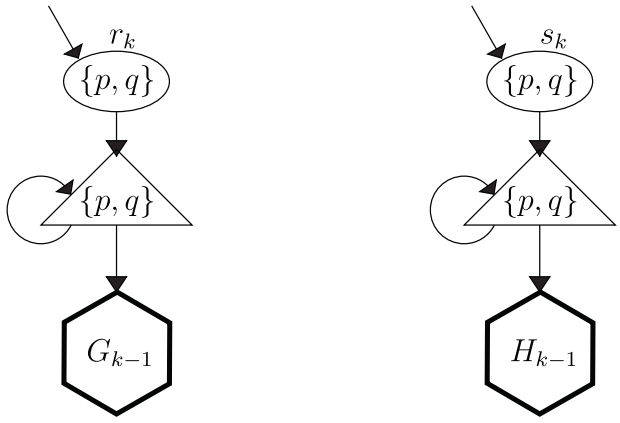
6.1. Comparison with GL

GL separates strategy quantifications from path quantifications. In comparison, ATL* and BSIL combine these two quantifications into SQs and SIQs. Thus, with GL, we can specify that, for all S-profiles of A, there exists a play satisfying 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 6.1 uses two inductive families of game graphs to show that GL is not as expressive as BSIL. The base cases, G_1 and H_1, are in Figure 6 for three agents. The inductive cases G_{k+1} and H_{k+1} are respectively constructed out of G_k and H_k as in Figure 7.

LEMMA 6.1. *Every GL formula ϕ with k ($k > 0$) SQs cannot distinguish G_k and H_k, while $\langle 1\rangle((\langle +2\rangle\Box p) \wedge \langle +2\rangle\Box q)$ can.*

PROOF. 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 no matter the value of k. The proof continues by an induction on k and the number of SQs in ϕ.



(a) G_k (b) H_k

◯ belongs to Agent 1; □ belongs to Agent 2; and △ belongs to Agent 3.

Fig. 7. Inductive cases for the expressiveness of BSIL over ATL*.

Base case: Assume that ϕ has only one modal operator. Then there are the following case analyses 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 6(a) if and only if there exists a strategy for A to characterize S in Figure 6(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 6, 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 6. 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 6. 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 cannot do it either.

Thus, the base case is proven.

Induction step: If ϕ has k modal operators, to tell the difference between G_k and H_k, we need a modal subformula of ϕ that can tell the difference between G_{k-1} and

H_{k-1}. But according to the inductive hypothesis, this is impossible. Thus, the lemma is proven. □

THEOREM 6.2. *GL formula* $\exists\{1\}.((\exists\Box p) \wedge \exists\Box q)$ *is not equivalent to any BSIL formula.*

PROOF. The proof basically follows the same argument in Alur et al. [2002] that $\exists\{1\}.((\exists\Box p) \wedge \exists\Box q)$ is not equivalent to any ATL* formula. □

Lemma 6.1 and Theorem 6.2 together show that GL and BSIL are not comparable in expressiveness.

6.2. Comparison with AMC

AMC is an extension from μ-calculus and allows for multiple fixpoints interleaved together [Alur et al. 2002]. 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 $\mathbf{lfp}x.\psi_1(x)$, 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 an even number of negations. The duality of the least fixpoint operator is the greatest fixpoint operator **gfp**. Formula $\mathbf{gfp}x.\psi_1$ is defined as $\neg\mathbf{lfp}x.\neg\psi_1(x)$.

To establish that AMC is not as expressive as BSIL, we basically follow the proof style for Lemma 6.8 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 $\mathbf{lfp}x.\phi$ in a given AMC formula share the same quantified name of x. Given a subformula $\mathbf{lfp}x_i.\phi$ 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 is 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 $\phi \in F_0(x)$, $\kappa(\mathcal{G}, \neg\phi) = \mathcal{Q} - \kappa(\mathcal{G}, \phi)$.
- 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 6.3. *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* $F_0(x)$, $r_0 \in \kappa(G_0, \phi_1)$ *if and only if* $s_0 \in \kappa(H_0, \phi_1)$ *in Figure 6.*

PROOF. We can prove this with a 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 cannot 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 $\text{AMC}^{(k)}$ 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 $\text{AMC}^{(2)}$. Then, $\text{AMC}^{(0)}$ is the smallest set with the following restrictions:

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

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

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

Note that there could be free variables in the formulas classified previously. 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 formula 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 Alur et al. [2002]. 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 6.4. *Assume that G_k and H_k are defined in Figures 6 and 7. For every k, AMC formula $\phi \in AMC^{(k)}$, and two consistent valuations ν and ν', respectively, of G_k and H_k, $r_k \in (\phi)^{G_k}(\nu)$ if and only if $s_k \in (\phi)^{H_k}(\nu')$.*

PROOF. We use an induction on k to prove the lemma.

Base case: When ϕ is in Cases 1a through 1d, the lemma follows straightforwardly. In Case 1e, $(\mathbf{lfp}x.\psi)^{G_1}(\nu)$ can be expanded as follows. We let

- $\psi^{G_1,\nu,(0)}$ be $(\psi[x \mapsto \mathit{false}])^{G_1}(\nu)$ and
- for each $h > 0$, $\psi^{G_1,\nu,(h)}$ be $(\phi[x \mapsto \psi_1^{G_1,(h-1)}])(\nu)$.

Then, $(\mathbf{lfp}x.\phi)^{G_1}(\nu) = \bigcup_{h \geq 0} \psi^{G_1,\nu,(h)}$ according to the semantics of AMC. Similarly, $(\mathbf{lfp}x.\phi)^{H_1}(\nu') = \bigcup_{h \geq 0} \psi^{H_1,\nu',(h)}$. According to the same argument for Cases 1a through 1d, for each $h \geq 0$, $r_0 \in \psi^{G_1,\nu,(h)}$ if and only if $s_0 \in \psi^{H_1,\nu',(h)}$. Thus, it is clear that $r_0 \in (\mathbf{lfp}x.\phi)^{G_1}(\nu)$ if and only if $s_0 \in (\mathbf{lfp}x.\phi)^{H_1}(\nu')$. 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 6.4, we conclude the proof for Lemma 6.5 in the following.

LEMMA 6.5. *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 6.4, it is apparent that ϕ with $\phi \in \mathrm{AMC}^{(k)}$ cannot tell G_k and H_k. □

By the same argument in Alur et al. [2002], for the one-agent game, BSIL coincides with CTL and is not as expressive as AMC.

THEOREM 6.6. *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 6.1 and 6.5 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.

6.3. Comparison with ATL*

It is easy to see that BSIL is a superclass of ATL. Thus, we have the following lemma.

LEMMA 6.7. *BSIL is at least as expressive as ATL.*

Then, Lemmas 6.1 and 6.5 lead to the fact that there are some BSIL properties that ATL* cannot express since GL and AMC are both superclasses of ATL* [Alur et al. 2002].

LEMMA 6.8. *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.*

Lemmas 6.7 and 6.8 together establish that ATL is strictly less expressive than BSIL. Then, the following lemma shows the reverse direction.

THEOREM 6.9. *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 [Alur et al. 2002]. □

Lemmas 6.8 and 6.9 together establish that ATL* and BSIL are not comparable in expressiveness.

7. 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 [Alur et al. 2002], we can evaluate the proper state subformulas 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.

As explained in Section 3, our algorithm also labels subformulas and their symbolic S-profiles on the nodes in a computation tree. The formula is satisfied if and only if we can find a finite tree top with labels consistent with the SQs and SIQs in the input

formula and can be extended to an infinite computation tree. In Section 7.1, we derive the labels (i.e., subformulas and their symbolic S-profiles) that are sufficient for our model-checking algorithm and present two procedures:

- `localEval()`, which checks whether the satisfaction of formulas at a state can be decided locally, and
- `sucSet()`, which nondeterministically chooses a scheme to pass down the subformulas and their symbolic S-profiles to the child nodes without violating the restrictions on the S-profiles declared with the SQs and SIQs.

There are, however, severe differences between exploring a computation tree for BSIL and exploring one for ATL [Alur et al. 2002]. 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 situations 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.

Then, in Section 7.2, we present our algorithm in two parts, one for model checking BSIL state formulas and the other for model checking BSIL tree formulas. In Section 7.3, we prove the correctness of the algorithm. In Section 7.4, we show that our algorithm is in PSPACE. Together with Lemma 7.8 in Section 7.5, we then establish the PSPACE-completeness of the BSIL and ATL$^+$ model-checking problems.

7.1. 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 $def(f)$. Inheriting the notations in Mogavero et al. [2010], we may also represent the the mapping as $(a_1, s_1)(a_2, s_2)\ldots(a_n, s_n)$.

We need some special techniques in checking tree formulas. We adopt the concept of strategy variables from Chatterjee et al. [2010] and Mogavero et al. [2010]. A *strategy variable binding* (*SV binding* for short) is a partial function from $[1, m]$ to strategy variables. Given an SV binding Λ, $\Lambda \circ (a_1, s_1)\ldots(a_n, s_n)$ 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 $a \in def(\Sigma_i) \cap def(\Sigma_j)$, $\Sigma_i(a) = \Sigma_j(a)$ 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 formulas 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 $((1, s_1)(2, s_2)\Diamond p) \wedge (1, s_1)(2, s_3)\Box q \vee \Diamond r$, 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.

Suppose we are given a function π that maps symbolic strategy names to strategies. Similar to the semantics of strategy logics [Mogavero et al. 2010] 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:

Table I. 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.

- $\mathcal{G}, q \models^{\pi} \Lambda_1\psi_1 \vee \Lambda_2\psi_2$ if and only if $\mathcal{G}, q \models^{\pi} \Lambda_1\phi_1$ or $\mathcal{G}, q \models^{\pi} \Lambda_2\phi_2$ holds.
- $\mathcal{G}, q \models^{\pi} \Lambda_1\phi_1 \wedge \Lambda_2\phi_2$ if and only if 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 $\Sigma = \{a \mapsto \pi(\Lambda(a)) \mid a \in def(\Lambda)\}$, $\mathcal{G}, q \models^{\pi} \Lambda\psi_1$ if and only if for all plays ρ compatible with Σ from q, $\rho \models_{\Sigma} \psi_1$ holds.

In Table I, 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 I with the semantics of BSIL formulas.

LEMMA 7.1. *Suppose we are given a state q, a BSIL subformula ψ, and an S-profile Σ such that $\mathcal{G}, q \models_{\Sigma} \psi$. There exist an SV binding Λ and a function π such that $\mathcal{G}, q \models^{\pi} bf(\Lambda\psi)$.*

PROOF. We construct Λ and π as follows. Without loss of generality, we assume ψ is unique in the input formula. For every $a \in def(\Sigma)$, we let $\Lambda(a) = s_a^{\psi}$ and $\pi(s_a^{\psi}) = \Sigma(a)$. It is clear that the functional composition of Λ and π is actually Σ. Then, according to the semantics of $\mathcal{G}, q \models^{\pi} bf(\Lambda\psi)$ presented earlier, $\mathcal{G}, q \models^{\pi} bf(\Lambda\psi)$ since for all play ρ compatible with Σ from q, $\rho \models_{\Sigma} \psi$. Thus, the lemma is proven. □

LEMMA 7.2. *Suppose we are given a state q, a BSIL subformula ψ, an SV binding Λ, and a function π such that $\mathcal{G}, q \models^{\pi} bf(\Lambda\psi)$. Then there exists an S-profile Σ such that $\mathcal{G}, q \models_{\Sigma} \psi$.*

PROOF. We can construct Σ by defining, for all $a \in def(\Lambda)$, $\Sigma(a) = \pi(\Lambda(a))$. Thus, Σ is the functional composition of Λ and π. Then, according to the semantics of $\mathcal{G}, q \models^{\pi} bf(\Lambda\psi)$ presented earlier, $\mathcal{G}, q \models^{\pi} bf(\Lambda\psi)$ implies $\mathcal{G}, q \models_{\Sigma} \psi$. Thus, the lemma is proven. □

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

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

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

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

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

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

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

There is a new S-profile introduced:

- Σ_3 for $(1,s_1)(2,s_2)(3,s_4)(4,s_6)$ 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 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:

`localEval`(W, θ) // $\lambda()$ 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 `localEval`(W, θ_1) and θ_2 be `localEval`(W, θ_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 `localEval`(W, θ_1) and θ_2 be `localEval`(W, θ_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** $\phi_1 \mathrm{U} \phi_2$**:**
18: **if** $\phi_2 \in W$ **then return** *true* **else if** $\phi_1 \notin P$ **then return** *false* **else return** θ **end if**
19: **end switch**

Statement 18 checks if $\Lambda\theta$ is fulfilled. When it is fulfilled, θ is changed to *true*. Statements 18 and 16 also check if $\Lambda\theta$ is violated. When a violation happens, θ is changed to *false*.

Then we need a procedure, `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. For example, `next`$(\bigcirc\phi) = \phi$, `next`$(\Diamond\phi) = \Diamond\phi$, `next`$(\phi_1 \mathrm{U} \phi_2) = \phi_1 \mathrm{U} \phi_2$, and `next`$(\Box\phi) = \Box\phi$.

With the two basic procedures defined earlier, 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$\{\Lambda$`localEval`$(\lambda(q), \theta) \mid \Lambda\theta \in C\}$.
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 $a \in def(\Lambda)$ with $\delta((q, q'), a) \neq \alpha_{\Lambda(a)}$ **then** return $\emptyset$ **end if**

8: **for** $(q', C') \in \Delta$ with $\forall a \in def(\Lambda)(\delta((q, q'), a) = \alpha_{\Lambda(a)})$ **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 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.

7.2. 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', ϕ_1) in the following that checks a BSIL property ϕ_1 at each state in Q':

checkSetOfBSIL(Q', ϕ_1)
1: **if** $\phi_1 \notin P \cup \{true, false\}$ **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 $\eta_1 \vee \cdots \vee \eta_n$.
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 $\texttt{recTree}(q, C)$ in the following, and pass down the path obligations to the children according to the restrictions of the SV binding in C:

$\texttt{recTree}(q, C)$

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** $\texttt{sucSet}(q, C)$ is empty **then return** *false* **end if**
5: **for** each $(q', C') \in \texttt{sucSet}(q, C)$ with $C' \neq \emptyset$ **do**
6: **if** $\texttt{recTree}(q', C')$ 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 $\texttt{sucSet}(q, C)$ to nondeterministically calculate an assignment of path obligations to the children of q.

7.3. 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 a 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:
  —$C = \beta(r)$.
  —For every $v \in V$ and every $(q', C') \in \texttt{sucSet}(\alpha(v), \beta(v))$, there exists a $(v, v') \in E$ with $\alpha(v') = q'$ and $\beta(v') = C'$.

The OD tree is *fulfilled* if and only if 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 neither $\Lambda \bigcirc \phi_1 \in \beta(v_j)$ nor $\Lambda\phi_1 \mathrm{U} \phi_2 \in \beta(v_j)$. 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 7.4. *For a set C of BP obligations,* $\texttt{recTree}(q, C)$ *returns true if and only if 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 leaves 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 $\Lambda \bigcirc \phi_1$ or $\Lambda\phi_1 \mathrm{U} \phi_2$ 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 suffixes 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 roots of T and T' have the same α and β labels. We can repeat this replacement until no node labeled with either a next-formula or 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 leaves, and returns *true*. □

LEMMA 7.5. *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 $\mathcal{G}, q \models^{\pi} \eta$ if and only if 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 7.4 and 7.5. 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 7.6. *Given a state q in $\mathcal{G}$, $\texttt{checkBSIL}(q, \chi)$ if and only if $\mathcal{G}, q \models_{\perp} \chi$.*

7.4. Complexities of the Algorithm

The algorithm that we presented in Sections 7.1 and 7.2 can run in PSPACE mainly because we can enumerate the conjunts in a disjunctive normal form (DNF) expression 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 $\texttt{sucSet}(q, C)$ to calculate the assignment of BP obligations to the children to q in the computation tree. Specifically, procedure $\texttt{sucSet}(q, C)$ nondeterministically returns a set Θ with elements of the form (q', C') such that $(q, q') \in R$ and $C' \subseteq C$ since in procedure $\texttt{sucSet}(q, C)$, a path obligation $\Lambda\theta$ is passed down to a child and recorded in the corresponding C' only when it matches the for loop condition at statement 8. 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 the 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 holds:

- 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 $\texttt{sucSet}(q, C)$, the recursive exploration of a path can grow no longer than $1+|C|\cdot|Q|$. This leads to the following lemma.

LEMMA 7.7. *The BSIL model-checking algorithm in Sections 7.1 and 7.2 is in PSPACE.*

PROOF. For convenience, we let $\#(\chi)$ be the number of modal formulas in χ. Following the argument from earlier, 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 Θ returned from procedure $\texttt{sucSet}(q, C)$. Procedure $\texttt{sucSet}(q, C)$ can be nondeterministically computed by randomly assigning the obligations in C to the successors of q and checking if the assignment satisfies the strategy interaction restriction of the BP obligations. Procedure $\texttt{checkBSIL}(q, \phi)$ can then be executed in space cubic in the size of ϕ for the Θs 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 lengths 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 labelings 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 $O(|\chi| 2^{|\chi|} |Q|^{|\chi| \cdot (2+|\chi| \cdot |Q|)} |\chi|^{2|\chi|})$.

7.5. 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 [Garey and Johnson 1979] to an ATL$^+$ model-checking problem for a two-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 $\eta \equiv \bigtriangledown_1 p_1 \ldots \bigtriangledown_l p_l (C_1 \wedge C_2 \wedge \cdots \wedge C_n)$ 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 $l_{k,1} \vee \cdots \vee l_{k,h_k}$, where, for each $j \in [1, h_k]$, $l_{k,j}$ is a *literal*, that is, 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 reformulated so 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_ps with the decisions in those Ω_ps. For example, the 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.*

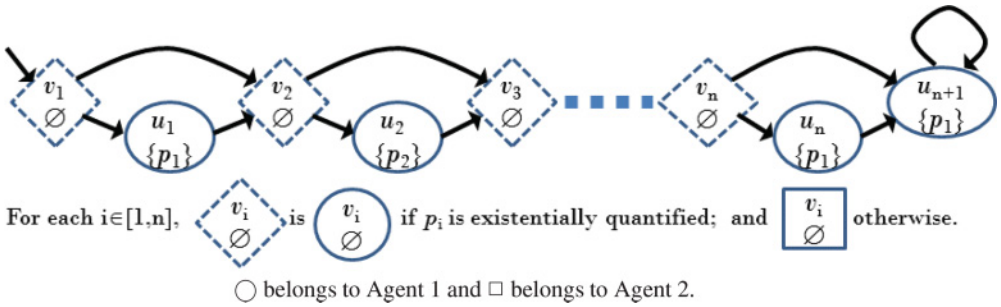


Fig. 8. Turn-based game graph for the PSPACE-hardness proof of a Boolean formula with n propositions.

The details of constructing those Γ_ps and Ω_ps can be found in the proof of the following lemma that establishes the PSPACE complexity lower bound.

LEMMA 7.8. *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 and "$\Box\neg p$" 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 8, 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.

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 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.

Given an $\eta = \langle 1\rangle \bigwedge_{1\le i\le k} \bigvee_{1\le j\le h_k} l_{i,j}$, we construct an ATL$^+$ formula ϕ_η as $\langle 1\rangle \bigwedge_{1\le i\le k} \bigvee_{1\le j\le h_k} \tau(l_{i,j})$, where $\tau(p) \stackrel{\text{def}}{=} \Diamond p$ and $\tau(\neg p) \stackrel{\text{def}}{=} \Box\neg p$; that is, 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 7.8.

Example 7.9. 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 7.8, 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 7.8 and 7.7, we obtain the complexity of our model-checking problems.

THEOREM 7.10. *The BSIL and ATL$^+$ model-checking problems are PSPACE-complete.*

8. AUTOMATA FOR BSIL MODEL CHECKING

In this section, we discuss a simple encoding of BSIL model checking in *alternating automata* on infinite trees [Kirsten 2002; Zappe 2002]. This naturally raises the question of 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 [Immerman 1981] 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.

8.1. Alternating Automata (AA)

Alternating automata (*AA*) are used to recognize ω-regular tree languages over labeled trees. Let $\mathbb{N}$ denote the set of nonnegative integers. Let $\mathbb{D} = [1, d]$ be an interval of $\mathbb{N}$. A $\mathbb{D}$ tree T is a nonempty 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 all elements in the set are interpreted true.

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

Definition 8.1. An alternating automaton A is a tuple $\langle X, U, u_0, \delta, \gamma \rangle$ with the following constraints:

- X is the set of labels of the analyzed trees.
- U is a finite set of states.
- $u_0 \in U$ is an initial state.

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

- $\delta : (U \times X) \mapsto \mathbb{B}^+(\mathbb{D} \times U)$ 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. (Please be reminded that $\mathbb{D}$ is an interval of $\mathbb{N}$ and represents a set of directions.)
- $\gamma : U \mapsto \mathbb{N}$ is a valuation function that labels each state with a nonnegative 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 $\xi'(\zeta' k) = (\zeta i, u')$.

It is easy to see that the out-degree of a node in T' is at most $d \cdot |U|$, while the out-degree of T is at most d.

Intuitively, we want to construct AAs with *states* that represent 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.

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 priority values 1 and 2 are used; that is, 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.

8.2. Model Checking with AAs

In this section, we relate the algorithms used for model checking from Section 7 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.

To keep the presentation simple, we study the algorithm for turn-based games and then turn to the general case of concurrent games.

8.2.1. Turn-Based Games. As in Section 7, 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 unraveling 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 labeled with a quadruple (a, k, S, L), containing the following information:

—$a \in [1, m]$ shows the owner of the node;
—$k \in \mathbb{D}$ shows the number of successors;
—$S \subseteq \mathbb{S}$ is the set of strategies, for which we can reach the node. The strategies in S are used just like atomic propositions; and
—H 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 labeling of states as reachable is consistent; that is, there are strategies in S such that for each strategy, exactly the nodes labeled by it are reachable.
(A2): For the respective set of paths described by this labeling, 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 unraveling of $\mathcal{G}$. However, the following observation obviously holds.

LEMMA 8.2. *$\mathcal{G}$ is a model of ϕ if and only if we can extend the unraveling 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 8.3. *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 recognizes 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 Π_a^k be the set of k tuples of disjoint sets $(S_a^1, S_a^2, S_a^3, \ldots, S_a^k)$ that cover[4] 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 $\delta_s(S, (a, k, S, H)) = \delta_k^s(a, S)$ and $\delta_s(S, (a, k, S', H)) = \mathit{false}$ 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 labeled with a strategy s_i, then no successor is labeled with s_i.
- If a state is labeled with a strategy s_i owned by a different agent than the node, then all successor states must be labeled with s_i.

[4]Note that disjoint cover does not mean partition, as we allow for empty sets.

- If a state is labeled with a strategy s_i owned by the same agent as the node, then exactly one successor state is labeled with s_i.

Also, any combination of the labels according to the last rule is possible. This exactly characterizes the labeling 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 8.4. *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 recognizes 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:

- $\delta_\wedge(C, (a, k, S, H)) = \mathit{false}$ 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.
- $\delta_\wedge(C, (a, k, S, H)) = \mathit{false}$ if there exists a $\Lambda\theta \in C$ with $\texttt{localEval}(H, \theta) = \mathit{false}$. Please recall that $\texttt{localEval}(H, \theta)$ converts a θ that is locally violated by H to *false*.
- Otherwise, $\delta_\wedge(C, (a, k, S, H)) = \bigwedge_{i \in [1,k]}(i, C')$ with $C' = \{\Lambda\texttt{next}(\texttt{localEval}(H, \theta)) \mid \Lambda\theta \in C\}$.

Note that $\mathcal{A}_\wedge$ is made deterministic by encoding the choices of strategy passing-down in the input symbol (a, k, S, H). Then, we define $\gamma_\wedge$ such that $\gamma_\wedge(C)$ is odd if and only if C contains an until-formula $\phi_1 \mathrm{U} \phi_2$ or a next-formula $\bigcirc\phi_1$. 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 8.5. *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 recognizes trees that satisfy (A1) and (A2).*

PROOF. The new state sets 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 $\delta_\wedge$ returns false and is applied independently for the two projections otherwise. Finally, $\gamma(u_s, u_\wedge) = \gamma_\wedge(u_\wedge)$. □

COROLLARY 8.6. *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 recognizes the trees with labeling functions that are projections from the trees that satisfy (A1) and (A2).*

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

The previous lemmas and corollaries make it easy to proof the main claim of this section.

THEOREM 8.7. *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 8.6 establishes this for BSIL 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 subformula ϕ in the specification by p_ϕ.
(3) We repeat the previous 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 one to run up to the number of disjuncts many times the respective construction 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 $\delta(u, (a, k, H)) = \bigvee_{S \subseteq \mathbb{S}} \delta'(u, (a, k, S, H))$.
- Let the acceptance and rejection player choose $\delta_s^k(a, S)$ 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 $I_i^l = [l, m]$ and $I_i^u = [m+1, u]$.
  (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 either with $I_{i+1} = I_i^l$ and $S_{i+1} = L_i$ or with $I_{i+1} = I_i^u$ and $S_{i+1} = U_i$.

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

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 transitions of $\mathcal{G}$. □

For PTIME-hardness, we reduce the reachability checking in AND/OR graphs [Immerman 1981] 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 8.8. *Model checking BSIL formulas is PTIME-complete in the size of the model.*

8.2.2. Concurrent Games. The differences that occur when studying the general case of concurrent game graphs rather than turn-based game structures are moderate.

Note that the weak deterministic AA for property (A2) from Lemma 8.4 is not affected by this change. The automaton that checks for property (A1), on the other hand, is affected.

To account for this change, we first revisit how the transition function of the automaton from the nondeterministic safety automaton in Lemma 8.3 works. The basic mechanism is a disjunction over the possible choices in Π_a^k, which represents the possible decisions made by the Agent a who owns the current position for his or her different strategies in S_a.

Having a concurrent game graph $A = \langle m, Q, r, P, \lambda, R, \Delta, \tau \rangle$, all agents need to make their respective decision. The strategies of each agent decide which token each agents selects.

We adjust the construction of the automaton that checks for property (A1) by using a not to name the agent, but to name the state (i.e., $a \in Q$), and assuming $Q = \{1, \ldots, k\}$, we redefine δ_s^k to

$$\delta_s^k(a, S) = \bigvee_{f:S \to \Delta} \bigwedge_{q \in Q} (q, S_q^f),$$

where $S_q^f \subseteq S$ is the subset of strategy names such that $s \in S_q^f$ if and only if

- $s \in S$ and
- $\tau((a, q), b) = f(s)$ holds for the Agent b for whom s is a strategy of Agent b.

f is simply a function that captures the decisions of the strategy.

The remainder of the constructions are not affected: the arguments used to establish Lemma 8.2, Corollaries 8.6 and 8.5, and Theorem 8.7 are not affected by this change.

Note that some information about the CGG is now incorporated into the automaton that checks (A1). The only minor precaution to be taken is that the number of decisions might be reduced for some agents in some states.

It is also possible to encode the transition function as part of the input letter instead. Note that, when model checking, this information is available through the CGG under consideration. The main theorem is therefore unaffected.

THEOREM 8.9. *Model checking BSIL formulas for CGGs is PTIME-complete in the size of the model.*

9. 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 [Muller and Schupp 1995] (or the automata constructions behind them [Safra 1988; Piterman 2007; Schewe 2009]), while hardness is a consequence of the inclusion of CTL$^+$ [Wilke 1999].

In the first step, we provide an AA for checking the consistency of a labeling as it is constructed in the model-checking approach in the previous section. This labeling 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 or concurrent 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 9.1. *We can build an AA $\mathcal{B}$ that is exponential in the size of a BSIL formula χ and accepts a fully labeled concurrent game graph if and only if the labeling is consistent and the concurrent 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 automaton 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 dualizing 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:

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

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

The argument only uses the automata from the previous section in the last line. Consequently, we can argue completely analogously for the concurrent game graphs.

LEMMA 9.2. *We can build an AA $\mathcal{B}$ that is exponential in the size of a BSIL formula χ and accepts a fully labeled turn-based game if and only if the labeling is consistent and the turn-based game is a model of χ.*

THEOREM 9.3. *The satisfiability of a BSIL formula ϕ can be checked in time doubly exponential in the size of φ. If φ is satisfiable by a turn-based game, 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 proceed 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 8.3) are sent to at most $|S_a| \leq |\mathbb{S}|$ directions. But successors of nodes to whom from no state a nonempty 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 of 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 for the number of required 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. □

Similarly, the particularities of a turn-based game are only used when determining the number of directions needed. The argument directly extends to the number of *decisions* needed in a concurrent game graph: the set Δ can be bounded accordingly, using the same argument. For m agents, we would then have $|\Delta|^m$ directions.

THEOREM 9.4. *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.*

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 cooperate, 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.

Table II. Experiment Data for the Prisoners' Dilemma Model

Prisoner Count		2	3	4	5	6	7
Property (A)	time	0.50s	0.92s	1.23s	3.19s	6.71s	25.s
	mem	72M	75M	83M	146M	388M	1043M
Property (B)	time	0.51s	0.88s	1.14s	2.90s	6.52s	23.1s
	mem	71M	75M	80M	140M	361M	979M

To obtain hardness, we exploit the 2EXPTIME-completeness of CTL$^+$. It is easy to see that CTL$^+$ is the sublogic 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$^+$ [Wilke 1999] provides us with:

COROLLARY 9.5. *Checking the realizability (both by a turn-based game and by a concurrent game graph) of a BSIL formula φ is 2EXPTIME-complete in the size of φ.*

10. EXPERIMENT

10.1. Implementation

We implemented a semisymbolic model checker of BSIL with **REDLIB** [Wang 2004, 2008a, 2008b, 2013, 2015], which is a free library for symbolic model checking based on decision diagrams. **REDLIB** supports symbolic precondition and postcondition calculation of discrete transitions. Moreover, the language to REDLIB supports flexible automata templates and powerful synchronization operators that allow for the concise modeling of multiparty synchronizations and synchronization correspondents with specific constraints.

Our model checker starts from a symbolic representation of the initial condition. Then, it repeatedly applies the postcondition procedure of **REDLIB** to explore the symbolic state representations in the computation tree. Our model checker uses the result in Section 7 to bound the exploration depth of the tree.

10.2. Benchmarks and Their Experiment Report

We experimented with three parameterized benchmarks to observe how our model-checking algorithm scales to the parameters. In all experiment data tables, we have the following notations: "s" denotes seconds for computation time, "M" denotes megabytes in memory usage, and "N/A" means "not available." All experiments are conducted on a PC with Intel i7-2600k 3.4GHZ CPU and 8G RAM running Ubuntu 12.04.

Prisoners' dilemma

The first is the prisoners' dilemma described in Example 1.1 with the number of prisoners as a parameter. Then we applied the model checker to check whether the model satisfies Formulas (A) and (B) in the introduction. The time and memory usage of the model checker are reported in Table II.

Banking systems

We also built a parameterized model for the banking system in our running example section. The parameter is the number of clients. We want to check the following three properties:

(I) The bank and the first client can together make a deposition transaction of the client successful. This property is satisfied.
(J) The bank and the first client can together make a transfer transaction from the client successful. The property is not satisfied since the partner bank of the transfer destination may not cooperate.
(K) The bank, the first client, and the partner can together make a transfer transaction successful. The property is satisfied.

Table III. Experiment Data for the Banking System Model

Client Counts		1	2	3	5	8	10	12
Property (I)	time	1.14s	3.21s	12.3s	103s	1417s	N/A	N/A
	mem	16M	19M	39M	92M	246M		
Property (J)	time	0.76s	0.84s	1.51s	3.18s	8.42s	26.1s	52.9s
	mem	18M	24M	31M	55M	99M	148M	245M
Property (K)	time	3.84s	12.5s	41.7s	1036s	N/A	N/A	N/A
	mem	21M	33M	67M	455M			

The time and memory usage of the model checker for the three properties are reported in Table III. It is easy to see that Formula (J) can be more efficiently checked than Formulas (I) and (K). The reason for the difference is that Formulas (I) and (K) are satisfied. Thus, we need to search for a tree top as a witness of the satisfaction. Usually, a satisfying tree top can be very deep and wide. Searching for such a tree top can be time-consuming and complex. In comparison, Formula (J) is not satisfied and can be refuted in a few steps of exploration from the root of the computation tree.

Election games

The third benchmark is for campaign strategies of two to four political parties 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 leaders of the parties who decide the amount of support that a candidate receives 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 nondeterministically.

In our model, in each round, the leaders concurrently make their budget allocation and then the election result is determined. For example, with \$3 in the budget of the leader of the first party with two candidates, the strategy of the leader 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) allocated to the first and the second candidates in the party. If there are only two parties, Leader 1 uses strategy (2, 1), and Leader 2 uses strategy (0, 2) (assume he or she has less budget), and the result of the round is that the two parties both win one seat in the round. On the other hand, if Leader 2 chooses strategy (1, 1) instead, then there is no strategy for Leader 1 to win more seats for his or her party.

We use the following seven properties to test our model checker:

(L) Party 1 has a strategy to make one of its candidate elected and allows Party 2 to win more seats than Party 1.
(M) Party 1 has a strategy to win more seats than Party 2 and allows Party 2 to make one of its candidates elected.
(N) Party 1 has a strategy to guarantee one of its candidates is elected and also to cooperate with Party 2 to prevent Party 3's candidates from being elected.
(O) Parties 1 and 2 can together prevent Party 3's candidates from being elected.
(P) Party 1 has a strategy to guarantee one of its candidates is elected, to cooperate with Party 2 to prevent Party 4's candidates from being elected, and to cooperate with Party 3 to prevent Party 5's candidates from being elected.
(Q) Party 1 has a strategy to cooperate with Party 2 to prevent Party 4's candidates from being elected and to cooperate with Party 3 to prevent Party 5's candidates from being elected.
(R) Party 1 has a strategy to guarantee one of its candidates is elected and to cooperate with Parties 2 and 3 to prevent Party 4's candidates from being elected.

Table IV. Experiment Data Election

Properties	Parameters									Result	Time	Mem
	s	c_1	b_1	c_2	b_2	c_3	b_3	c_4	b_4			
(L)	1	2	2	2	4	0	0	0	0	UNSAT	0.52s	61M
	2	2	4	2	4	0	0	0	0	UNSAT	1.20s	75M
	3	2	4	2	4	0	0	0	0	SAT	0.73s	75M
	3	3	6	3	6	0	0	0	0	SAT	1.51s	211M
	3	3	9	3	6	0	0	0	0	UNSAT	9.14s	837M
	3	3	6	3	9	0	0	0	0	SAT	8.74s	502M
	4	3	9	3	9	0	0	0	0	SAT	158s	6631M
(M)	1	2	2	2	4	0	0	0	0	UNSAT	0.53s	61M
	2	2	4	2	4	0	0	0	0	UNSAT	1.15s	75M
	3	2	4	2	4	0	0	0	0	UNSAT	1.14s	75M
	3	3	6	3	6	0	0	0	0	UNSAT	0.93s	211M
	3	3	9	3	6	0	0	0	0	SAT	5.22s	681M
	3	3	6	3	9	0	0	0	0	UNSAT	8.49s	502M
	4	3	9	3	9	0	0	0	0	UNSAT	93.17s	5082M
(N)	2	2	6	2	3	2	2	0	0	UNSAT	2.86s	493M
(O)										SAT	4.61s	368M
(P)	3	2	6	2	3	2	3	2	2	UNSAT	179s	3755M
(Q)										SAT	209s	2862M
(R)										SAT	75s	1329M

Parameters: c_i, b_i: # candidates and total budget of party i, respectively; s: # of seats in the congress.

The satisfaction of the properties may depend on the budgets and numbers of candidates of the parties. The seven properties allow us to observe how our algorithm works for different combinations of parameter values. The time and memory usage data for checking the seven properties are in Table IV.

10.3. Discussion of the Experiments

The experiment shows some possibilities of using our tool to flexibly support the analysis and synthesis of collaborating strategies among several agents. For the prisoners' dilemma and the banking system, the prisoners and the clients are respectively modeled as individual agents. In the experiment with the election game, the chairpersons of the parties are modeled as individual agents and their candidates and respective allocated budgets are modeled as numbers. It would be interesting to see how we can explore the techniques in modeling and specifying concurrent games with our tool.

On the verification side, we can see that the CPU time and memory usage exhibit typical combinatorial explosion for tools for solving difficult problems. As a preliminary tool, our experiment and implementation do shed light for performance enhancement research in the future. Specifically, our search for the OD trees does not take the shape of the game graphs and the BSIL formulas into account. It is possible to design heuristics that utilize the syntax information of the graphs and the formulas to construct OD trees.

11. CONCLUSION

BSIL can be useful in describing combinations of strategy profiles in a multiagent system. It is carefully designed for low model-checking cost while maintaining sufficient expressiveness to specify some useful properties. We have thoroughly investigated the theoretical aspects of BSIL, including the expressive power, the model-checking complexity, and the complexity of the satisfiability problem. Our experiment report also points out new research directions, including further extension to BSIL and performance-enhancing techniques to our model-checking algorithms.

ACKNOWLEDGMENTS

The authors would like to thank Professor Moshe Vardi, Professor Fang Yu, and the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. September 2002. Alternating-time temporal logic. *Journal of the ACM (JACM)* 49, 5 (Sept. 2002), 672–713.

Christel Baier, Tomas Brázdil, Marcus Gröser, and Antonin Kucera. 2007. Stochastic game logic. In *QEST*. IEEE Computer Society, 227–236.

Mordechai Ben-Ari, Amir Pnueli, and Zohar Manna. 1983. The temporal logic of branching time. *Acta Informatica* 20 (1983), 207–226.

Thomas Brihaye, Arnaud Da Costa, François Laroussinie, and Nicolas Markey. 2009. ATL with strategy contexts and bounded memory. In *LFCS*, Vol. LNCS 5407. Springer-Verlag, 92–106.

Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. 2010. Strategy logic. *Information and Computation* 208 (2010), 677–693.

Edmumd M. Clarke, Ernest Allen Emerson, and A. Prassad Sistla. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 8, 2 (1986), 244263.

Arnaud Da Costa, François Laroussinie, and Nicolas Markey. 2010. ATL with strategy contexts: Expressiveness and model checking. In *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 8. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 120–132.

E. Allen Emerson and Edmund M. Clarke. 1980. Characterizing correctness properties of parallel programs as fixpoints. In *Proceedings of the 7th Colloquium on Automata, Language, and Programming*. LNCS, vol. 85. Springer-Verlag.

E. Allen Emerson and Joe Y. Halpern. Feb. 1985. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences* 30, 1 (Feb. 1985), 1–24.

E. Allen Emerson and Joe Y. Halpern. Jan. 1986. 'Sometimes' and 'Not Never' Revisited: On branching versus linear time temporal logic. *Journal of ACM* 33, 1 (Jan. 1986), 151–178.

E. Allen Emerson and Chin-Laung Lei. 1987. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming* 8 (1987), 275–306.

Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Company.

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

Daniel Kirsten. 2002. Alternating tree automata and parity games. In *Automata, Logics, and Infinite Games*, Erich Gradel, Wolfgang Thomas, and Thomas Wilke (Eds.). LNCS, vol. 2500. Springer, 153–167.

Orna Kupferman, Parthasarathy Madhusudan, P. S. Thiagarajan, and Moshe Y. Vardi. 2000. Open systems in reactive environments: Control and synthesis. In *Proceedings of the 11th International Conference on Concurrency Theory.* LNCS, vol. 1877. Springer-Verlag, 92–107.

Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. 2001. Module checking. *Information and Computation* 164, 2 (2001), 322–344.

Leslie Lamport. 1980. Sometimes is sometimes "Not Never"-on the temporal logic of programs. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*. 174–185.

Franois Laroussinie and Nicolas Markey. 2013. Satisfiability of ATL with strategy contexts. In *Proceedings of the Workshop on Games, Automata, Logics and Formal Verification (GANDALF'13).* EPTCS, vol. 119. 208–223.

Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. 2012. What makes ATL* decidable? A decidable fragment of strategy logic. In *Concurrency Theory (CONCUR'12)*. LNCS, vol. 7454. Springer-Verlag, 193–208.

Fabio Mogavero, Aniello Murano, and Luigi Sauro. 2013. On the boundary of behavioral strategies. In *ACM/IEEE LICS*. 263–272.

Fabio Mogavero, Aniello Murano, and Luigi Sauro. 2014. A behavioral hierarchy of strategy logic. In *Computational Logic in Multi-Agent Systems*. LNCS, vol. 8624. 148–165.

Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. 2010. Reasoning about strategies. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*

(Leibniz International Proceedings in Informatics (LIPIcs)), Vol. 8. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 133–144.

David E. Muller and Paul E. Schupp. 1995. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science* 141, 1–2 (1995), 69–107.

Sophie Pinchinat. 2007. A generic constructive solution for concurrent games with expressive constraints on strategies. In *Automated Technology for Verification and Analysis (ATVA'07)*, Vol. LNCS 4762. Springer-Verlag, 253–267.

Nir Piterman. 2007. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Journal of Logical Methods in Computer Science* 3, 3 (2007).

Shmuel Safra. 1988. On the complexity of ω-automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS'88)*. IEEE Computer Society Press, 319–327.

Sven Schewe. 2009. Tighter bounds for the determinisation of Büchi automata. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'09)*. LNCS, vol. 5504. Springer-Verlag, 167–181.

Sven Schewe and Bernd Finkbeiner. 2007. Semi-automatic distributed synthesis. *International Journal of Foundations of Computer Science* 18, 1 (2007), 113–138.

Aravinda Prasad Sistla and Edmund M. Clarke. 1985. The complexity of propositional linear temporal logics. *Journal of the ACM (JACM)* 32, 3 (July 1985), 733–749.

Lawrence J. Stockmeyer. 1974. *The complexity of decision problems in automata theory and logic*. MIT.

Farn Wang. 2004. Efficient verification of timed automata with BDD-like data-structures. *International Journal of Software Tools for Technology Transfer (STTT)* 6, 1 (2004). Special issue for the *4th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, LNCS, vol. 2575, Springer-Verlag.

Farn Wang. 2008a. Efficient model-checking of dense-time systems with time-convexity analysis. In *IEEE Real-Time System Symposium (RTSS)*. IEEE Computer Society.

Farn Wang. 2008b. Time-progress evaluation for dense-time automata with concave path conditions. In *Automated Technology for Verification and Analysis (ATVA)*, Vol. LNCS 5311. Springer-Verlag.

Farn Wang. 2013. Efficient model-checking of dense-time systems with time-convexity analysis. *Theoretical Computer Science (TCS)* 467 (Jan. 2013), 89–108.

Farn Wang. 2015. Model-checking fair dense-time systems with propositions and events. *International Journal on Software Tools for Technology Transfer (STTT)* 17, 2 (2015), 223–243.

Thomas Wilke. 1999. CTL^+ is exponentially more succinct than CTL. In *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Springer-Verlag, 110–121.

Júlia Zappe. 2002. Modal μ-calculus and alternating tree automata. In *Automata, Logics, and Infinite Games*, Erich Gradel, Wolfgang Thomas, and Thomas Wilke (Eds.). LNCS, vol. 2500. Springer, 171–184.

Received February 2013; revised August 2014; accepted February 2015