Strategy Logic

Krishnendu Chatterjee[1], Thomas A. Henzinger[1,2], and Nir Piterman[2]

[1] University of California, Berkeley, USA
[2] EPFL, Switzerland
c_krish@eecs.berkeley.edu, {tah,Nir.Piterman}@epfl.ch

Abstract. We introduce *strategy logic*, a logic that treats strategies in two-player games as explicit first-order objects. The explicit treatment of strategies allows us to specify properties of nonzero-sum games in a simple and natural way. We show that the one-alternation fragment of strategy logic is strong enough to express the existence of Nash equilibria and secure equilibria, and subsumes other logics that were introduced to reason about games, such as ATL, ATL*, and game logic. We show that strategy logic is decidable, by constructing tree automata that recognize sets of strategies. While for the general logic, our decision procedure is nonelementary, for the simple fragment that is used above we show that the complexity is polynomial in the size of the game graph and optimal in the size of the formula (ranging from polynomial to 2EXPTIME depending on the form of the formula).

1 Introduction

In *graph games*, two players move a token across the edges of a graph in order to form an infinite path. The vertices are partitioned into player-1 and player-2 nodes, depending on which player chooses the successor node. The objective of player 1 is to ensure that the resulting infinite path lies inside a given winning set Ψ_1 of paths. If the game is zero-sum, then the goal of player 2 is to prevent this. More generally, in a nonzero-sum game, player 2 has her own winning set Ψ_2.

Zero-sum graph games have been widely used in the synthesis (or control) of reactive systems [22, 24], as well as for defining and checking the realizability of specifications [1, 8], the compatibility of interfaces [7], simulation relations between transition systems [11, 19], and for generating test cases [3], to name just a few of their applications. The study of nonzero-sum graph games has been more recent, with assume-guarantee synthesis [4] as one of its applications.

The traditional formulation of graph games consists of a two-player graph (the "arena") and winning conditions Ψ_1 and Ψ_2 for the two players (in the zero-sum case, $\Psi_1 = \neg\Psi_2$), and asks for computing the winning sets W_1 and W_2 of vertices for the two players (in the zero-sum case, determinacy [18] ensures that $W_1 = \neg W_2$). To permit the unambiguous, concise, flexible, and structured expression of problems and solutions involving graph games, researchers have introduced *logics* that are interpreted over two-player graphs. An example is the temporal logic ATL [2], which replaces the unconstrained path quantifiers of CTL with constrained path quantifiers: while the CTL formula $\forall\Psi$ asserts that the path property Ψ is inevitable —i.e., Ψ holds on all paths from a given state— the ATL formula $\langle\langle 1\rangle\rangle\Psi$ asserts that Ψ is enforcible by player 1 —i.e., player 1 has a strategy so that Ψ holds on all paths that can result from playing that strategy. The logic ATL has proved useful for expressing proof obligations in system verification, as well as for expressing subroutines of verification algorithms.

However, because of limitations inherent in the definition of ATL, several extensions have been proposed [2], among them the temporal logic ATL*, the alternating-time μ-calculus, and a so-called *game logic* of [2]: these are motivated by expressing general ω-regular winning conditions, as well as tree properties of computation trees that result from fixing the strategy of one player (module checking [17]). All of these logics treat strategies implicitly through modalities. This is convenient for zero-sum games, but awkward for nonzero-sum games. Indeed, it was not known if Nash equilibria, one of the most fundamental concepts in game theory, can be expressed in these logics.

In order to systematically understand the expressiveness of game logics, and to specify nonzero-sum games, we study in this paper a logic that treats strategies as explicit first-order objects. For example, using explicit strategy quantifiers, the ATL formula $\langle\langle 1\rangle\rangle\Psi$ becomes $(\exists x \in \Sigma)(\forall y \in \Gamma)\Psi(x,y)$ —i.e., "there exists a player-1 strategy x such that for all player-2 strategies y, the unique infinite path that results from the two players following the strategies x and y satisfies the property Ψ." Strategies are a natural primitive when talking about games and winning, and besides ATL and its extensions, Nash equilibria are naturally expressible in *strategy logic*.

As an example, we define *winning secure equilibria* [5] in strategy logic. A winning secure equilibrium is a special kind of Nash equilibrium, which is important when reasoning about the components of a system, each with its own specification. At such an equilibrium, both players can collaborate to satisfy the combined objective $\Psi_1 \wedge \Psi_2$. Moreover, whenever player 2 decides to abandon the collaboration and enforce $\neg\Psi_1$, then player 1 has the ability to retaliate and enforce $\neg\Psi_2$; that is, player 1 has a winning strategy for the relativized objective $\Psi_2 \Rightarrow \Psi_1$ (where $\Rightarrow$ denotes implication). The symmetric condition holds for player 2; in summary: $(\exists x \in \Sigma)(\exists y \in \Gamma)[(\Psi_1 \wedge \Psi_2)(x,y) \;\wedge\; (\forall y' \in \Gamma)(\Psi_2 \Rightarrow \Psi_1)(x,y') \;\wedge\; (\forall x' \in \Sigma)(\Psi_1 \Rightarrow \Psi_2)(x',y)]$. Note that the same player-1 strategy x which is involved in producing the outcome $\Psi_1 \wedge \Psi_2$ must be able to win for $\Psi_2 \Rightarrow \Psi_1$; such a condition is difficult to state without explicit quantification over strategies.

Our results are twofold. First, we study the expressive power of strategy logic. We show that the logic is rich enough to express many interesting properties of zero-sum and nonzero-sum games that we know, including ATL*, game logic (and thus module checking), Nash equilibria, and secure equilibria. Indeed, ATL* and the equilibria can be expressed in a simple fragment of strategy logic with no more than one quantifier alternation (note the $\exists\forall$ alternation in the above formula for defining winning secure equilibria). We also show that the simple one-alternation fragment can be translated to ATL* (the translation in general is double exponential in the size of the formula) and thereby the equilibria can be expressed in ATL^*.

Second, we analyze the computational complexity of strategy logic. We show that, provided all winning conditions are specified in linear temporal logic (or by word automata), strategy logic is decidable. The proof goes through automata theory, using tree automata to specify the computation trees that result from fixing the strategy of one player. The complexity is nonelementary, with the number of exponentials depending on the quantifier alternation depth of the formula. In the case of the simple one-alternation fragment of strategy logic, which suffices to express ATL^* and equilibria, we obtain much better bounds: for example, for infinitary path formulas (path formulas that are independent of finite prefixes), there is a linear translation of a simple one-alternation fragment formula to an ATL^* formula.

In summary, strategy logic provides a decidable language for talking in a natural and uniform way about all kinds of properties on game graphs, including zero-sum, as well as nonzero-sum objectives. Of course, for more specific purposes, such as zero-sum reachability games, more restrictive and less expensive logics, such as ATL, are more appropriate; however, the consequences of such restrictions, and their relationships, is best studied within a clean, general framework such as the one provided by strategy logic. In other words, strategy logic can play for reasoning about games the same role that first-order logic with explicit quantification about time has played for temporal reasoning: the latter has been used to categorize and compare temporal logics (i.e., logics with implicit time), leading to a notion of completeness and other results in correspondence theory [10, 15].

In this work we consider perfect-information games and, consequently, only pure strategies (no probabilistic choice). An extension of this work to the setting of partial-information games is an interesting research direction (cf. [12]). Other possible extensions include reasoning about concurrent games and about perfect-information games with probabilistic transitions, as well as increasing the expressive power of the logic by allowing more ways to bound strategies (e.g., comparing strategies).

2 Graph Games

A *game graph* $G = ((S, E), (S_1, S_2))$ consists of a directed graph (S, E) with a finite set S of states, a set E of edges, and a partition (S_1, S_2) of the state space S. The states in S_1 are called player-1 states; the states in S_2, player-2 states. For a state $s \in S$, we write $E(s)$ to denote the set $\{t \mid (s, t) \in E\}$ of successor states. We assume that every state has at least one out-going edge; i.e., $E(s)$ is nonempty for all $s \in S$.

Plays. A game is played by two players: player 1 and player 2, who form an infinite path in the game graph by moving a token along edges. They start by placing the token on an initial state and then they take moves indefinitely in the following way. If the token is on a state in S_1, then player 1 moves the token along one of the edges going out of the state. If the token is on a state in S_2, then player 2 does likewise. The result is an infinite path $\pi = \langle s_0, s_1, s_2, \ldots \rangle$ in the game graph; we refer to such infinite paths as plays. Hence given a game graph G, a *play* is an infinite sequence $\langle s_0, s_1, s_2, \ldots \rangle$ of states such that for all $k \geq 0$, we have $(s_k, s_{k+1}) \in E$. We write Π for the set of all plays.

Strategies. A strategy for a player is a recipe that specifies how to extend plays. Formally, a *strategy* σ for player 1 is a function σ: $S^* \cdot S_1 \to S$ that given a finite sequence of states, which represents the history of the play so far, and which ends in a player-1 state, chooses the next state. A strategy must choose only available successors, i.e., for all $w \in S^*$ and all $s \in S_1$, we have $\sigma(w \cdot s) \in E(s)$. The strategies for player 2 are defined symmetrically. We denote by Σ and Γ the sets of all strategies for player 1 and player 2, respectively. Given a starting state $s \in S$, a strategy σ for player 1, and a strategy τ for player 2, there is a unique play, denoted as $\pi(s, \sigma, \tau) = \langle s_0, s_1, s_2, \ldots \rangle$, which is defined as follows: $s = s_0$, and for all $k \geq 0$, we have (a) if $s_k \in S_1$, then $\sigma(s_0, s_1, \ldots, s_k) = s_{k+1}$, and (b) if $s_k \in S_2$, then $\tau(s_0, s_1, \ldots, s_k) = s_{k+1}$.

3 Strategy Logic

Strategy logic is interpreted over labeled game graphs. Let P be a finite set of atomic propositions. A *labeled game graph* $\mathcal{G} = (G, P, L)$ consists of a game graph G together with a labeling function L: $S \to 2^P$ that maps every state s to the set $L(s)$ of atomic propositions that are true at s. We assume that there is a special atomic proposition $\mathbf{tt} \in P$ such that $\mathbf{tt} \in L(s)$ for all $s \in S$.

Syntax. The formulas of strategy logic consist of the following kinds of subformulas. Path formulas Ψ are LTL formulas, which are interpreted over infinite paths of states. Atomic strategy formulas are path formulas $\Psi(x, y)$ with two arguments —a variable x that denotes a player-1 strategy, and a variable y that denotes a player-2 strategy. From atomic strategy formulas, we define a first-order logic of quantified strategy formulas. The formulas of strategy logic are the closed strategy formulas (i.e., strategy formulas without free strategy variables); they are interpreted over states. We denote path and strategy formulas by Ψ and Φ, respectively. We use the variables $x, x_1, x_2, \ldots$ to range over strategies for player 1, and denote the set of such variables by X; similarly, the variables $y, y_1, y_2, \ldots \in Y$ range over strategies for player 2. Formally, the path and strategy formulas are defined by the following grammar:

$$\Psi ::= p \mid \Phi \mid \Psi \wedge \Psi \mid \neg\Psi \mid \bigcirc\Psi \mid \Psi \,\mathcal{U}\, \Psi, \text{ where } p \in P \text{ and } \Phi \text{ is closed;}$$

$$\Phi ::= \Psi(x, y) \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid Qx.\Phi \mid Qy.\Phi, \text{ where } Q \in \{\exists, \forall\}, x \in X, y \in Y.$$

Observe that the closed strategy formulas can be reused as atomic propositions. We formally define the free variables of strategy formulas as follows:

$\mathsf{Free}(\Psi(x, y)) = \{x, y\}$;
$\mathsf{Free}(\Phi_1 \wedge \Phi_2) = \mathsf{Free}(\Phi_1) \cup \mathsf{Free}(\Phi_2)$;

$\mathsf{Free}(\Phi_1 \vee \Phi_2) = \mathsf{Free}(\Phi_1) \cup \mathsf{Free}(\Phi_2)$;
$\mathsf{Free}(Qx.\Phi') = \mathsf{Free}(\Phi') \setminus \{x\}$, for $Q \in \{\exists, \forall\}$;
$\mathsf{Free}(Qy.\Phi') = \mathsf{Free}(\Phi') \setminus \{y\}$, for $Q \in \{\exists, \forall\}$.

A strategy formula Φ is *closed* if $\mathsf{Free}(\Phi) = \emptyset$. We define additional boolean connectives such as $\Rightarrow$, and additional temporal operators such as $\Box$ and $\Diamond$, as usual.

Semantics. For a set $Z \subseteq X \cup Y$ of variables, a *strategy assignment* A_Z assigns to every variable $x \in Z \cap X$, a player-1 strategy $A_Z(x) \in \Sigma$, and to every variable $y \in Z \cap Y$, a player-2 strategy $A_Z(y) \in \Gamma$. Given a strategy assignment A_Z and player-1 strategy $\sigma \in \Sigma$, we denote by $A_Z[x \leftarrow \sigma]$ the extension of the assignment A_Z to the set $Z \cup \{x\}$, defined as follows: for $w \in Z \cup \{x\}$, we have $A_Z[x \leftarrow \sigma](w) = A_Z(w)$ if $w \neq x$, and $A_Z[x \leftarrow \sigma](x) = \sigma$. The definition of $A_Z[y \leftarrow \tau]$ for player-2 strategies $\tau \in \Gamma$ is analogous.

The semantics of path formulas Ψ is the usual semantics of LTL. We now describe the satisfaction of a strategy formula Φ at a state $s \in S$ with respect to a strategy assignment A_Z, where $\mathsf{Free}(\Phi) \subseteq Z$:

$$\begin{array}{lll}
(s, A_Z) \models \Psi(x, y) & \text{iff} & \pi(s, A_Z(x), A_Z(y)) \models \Psi; \\
(s, A_Z) \models \Phi_1 \wedge \Phi_2 & \text{iff} & (s, A_Z) \models \Phi_1 \text{ and } (s, A_Z) \models \Phi_2; \\
(s, A_Z) \models \Phi_1 \vee \Phi_2 & \text{iff} & (s, A_Z) \models \Phi_1 \text{ or } (s, A_Z) \models \Phi_2; \\
(s, A_Z) \models \exists x.\Phi' & \text{iff} & \exists \sigma \in \Sigma.\ (s, A_Z[x \leftarrow \sigma]) \models \Phi'; \\
(s, A_Z) \models \forall x.\Phi' & \text{iff} & \forall \sigma \in \Sigma.\ (s, A_Z[x \leftarrow \sigma]) \models \Phi'; \\
(s, A_Z) \models \exists y.\Phi' & \text{iff} & \exists \tau \in \Gamma.\ (s, A_Z[y \leftarrow \tau]) \models \Phi'; \\
(s, A_Z) \models \forall y.\Phi' & \text{iff} & \forall \tau \in \Gamma.\ (s, A_Z[y \leftarrow \tau]) \models \Phi'.
\end{array}$$

The semantics of a closed strategy formula Φ is the set $[\![\Phi]\!] = \{s \in S \mid (s, A_\emptyset) \models \Phi\}$ of states.

Unnested path formulas. Of special interest is the fragment of strategy logic where path formulas do not allow any nesting of temporal operators. This fragment has a CTL-like flavor, and as we show later, results in a decision procedure with a lower computational complexity. Formally, the *unnested* path formulas are restricted as follows:

$$\Psi ::= p \mid \Phi \mid \Psi \wedge \Psi \mid \neg \Psi \mid \bigcirc \Phi \mid \Phi\ \mathcal{U}\ \Phi, \text{ where } p \in P \text{ and } \Phi \text{ is closed.}$$

The resulting closed strategy formulas are called the *unnested-path-formula fragment* of strategy logic.

Examples. We now present some examples of formulas of strategy logic. We first show how to express formulas of the logics ATL and ATL^* [2] in strategy logic. The alternating-time temporal logic ATL^* consists of path formulas quantified by the alternating path operators $\langle\!\langle 1 \rangle\!\rangle$ and $\langle\!\langle 2 \rangle\!\rangle$, the existential path operator $\langle\!\langle 1, 2 \rangle\!\rangle$ (or $\exists$), and the universal path operator $\langle\!\langle \emptyset \rangle\!\rangle$ (or $\forall$). The logic ATL is the subclass of ATL^* where only unnested path formulas are considered. Some examples of ATL and ATL^* formulas and the equivalent strategy formulas are as follows: for a proposition $p \in P$,

$$\begin{aligned}
\langle\!\langle 1 \rangle\!\rangle(\Diamond p) &= \{s \in S \mid \exists \sigma.\ \forall \tau.\ \pi(s,\sigma,\tau) \models \Diamond p\} = [\![\exists x.\ \forall y.\ (\Diamond p)(x,y)]\!];\\
\langle\!\langle 2 \rangle\!\rangle(\Box\Diamond p) &= \{s \in S \mid \exists \tau.\ \forall \sigma.\ \pi(s,\sigma,\tau) \models \Box\Diamond p\} = [\![\exists y.\ \forall x.\ (\Box\Diamond p)(x,y)]\!];\\
\langle\!\langle 1,2 \rangle\!\rangle(\Box p) &= \{s \in S \mid \exists \sigma.\ \exists \tau.\ \pi(s,\sigma,\tau) \models \Box p\} = [\![\exists x.\ \exists y.\ (\Box p)(x,y)]\!];\\
\langle\!\langle \emptyset \rangle\!\rangle(\Diamond\Box p) &= \{s \in S \mid \forall \sigma.\ \forall \tau.\ \pi(s,\sigma,\tau) \models \Box p\} = [\![\forall x.\ \forall y.\ (\Diamond\Box p)(x,y)]\!].
\end{aligned}$$

Consider the strategy formula $\Phi = \exists x.\ (\exists y_1.\ (\Box p)(x,y_1)\ \wedge\ \exists y_2.\ (\Box q)(x,y_2))$. This formula is different from the two formulas $\langle\!\langle 1,2 \rangle\!\rangle(\Box p) \wedge \langle\!\langle 1,2 \rangle\!\rangle(\Box q)$ (which is too weak) and $\langle\!\langle 1,2 \rangle\!\rangle(\Box(p \wedge q))$ (which is too strong). It follows from the results of [2] that the formula Φ cannot be expressed in ATL^*.

One of the features of strategy logic is that we can restrict the kinds of strategies that interest us. For example, the following strategy formula describes the states from which player 1 can ensure the goal Φ_1 while playing against any strategy that ensures Φ_2 for player 2:

$$\exists x_1.\ \forall y_1.\ ((\forall x_2. \Phi_2(x_2,y_1))\ \Rightarrow\ \Phi_1(x_1,y_1))$$

The mental exercise of "I know that you know that I know that you know ..." can be played in strategy logic up to any constant level. The analogue of the above formula, where the level of knowledge is nested up to level k, can be expressed in strategy logic. For example, the formula above ("knowledge nesting 1") is different from the following formula with "knowledge nesting 2":

$$\exists x_1.\ \forall y_1.\ ((\forall x_2.(\forall y_2. \Phi_1(x_2,y_2))\ \Rightarrow\ \Phi_2(x_2,y_1))\ \Rightarrow\ \Phi_1(x_1,y_1))$$

We do not know whether the corresponding fixpoint of 'full knowledge nesting' can be expressed in strategy logic.

As another example, we consider the notion of dominating and dominated strategies [21]. Given a path formula Ψ and a state $s \in S$, a strategy x_1 for player 1 *dominates* another player-1 strategy x_2 if for all player-2 strategies y, whenever $\pi(s,x_2,y) \models \Psi$, then $\pi(s,x_1,y) \models \Psi$. The strategy x_1 is *dominating* if it dominates every player-1 strategy x_2. The following strategy formula expresses that x_1 is a dominating strategy:

$$\forall x_2.\ \forall y.\ (\Psi(x_2,y)\ \Rightarrow\ \Psi(x_1,y))$$

Given a path formula Ψ and a state $s \in S$, a strategy x_1 for player 1 is *dominated* if there is a player-1 strategy x_2 such that (a) for all player-2 strategies y_1, if $\pi(s,x_1,y_1) \models \Psi$, then $\pi(s,x_2,y_1) \models \Psi$, and (b) for some player-2 strategy y_2, we have both $\pi(s,x_2,y_2) \models \Psi$ and $\pi(s,x_1,y_2) \not\models \Psi$. The following strategy formula expresses that x_1 is a dominated strategy:

$$\exists x_2.\ ((\forall y_1.\ \Psi(x_1,y_1)\ \Rightarrow\ \Psi(x_2,y_1))\ \wedge\ (\exists y_2.\ \Psi(x_2,y_2) \wedge \neg\Psi(x_1,y_2)))$$

The formulas for dominating and dominated strategies express properties about strategies and are not closed formulas.

4 Simple One-Alternation Fragment of Strategy Logic

In this section we define a subset of strategy logic. Intuitively, the alternation depth of a formula is the number of changes between $\exists$ and $\forall$ quantifiers (a formal definition is given in Section 6). The subset we consider here is a subset of the formulas that allow only one alternation of strategy quantifiers. We refer to this subset as the simple one-alternation fragment. We show later how several important concepts in nonzero-sum games can be captured in this fragment.

Syntax. We are interested in strategy formulas that depend on three path formulas: Ψ_1, Ψ_2, and Ψ_3. The strategy formulas in the simple one-alternation fragment assert that there exist player-1 and player-2 strategies that ensure Ψ_1 and Ψ_2, respectively, and at the same time cooperate to satisfy Ψ_3. Formally, the *simple one-alternation* strategy formulas are restricted as follows:

$$\Phi ::= \Phi \wedge \Phi \mid \neg\Phi \mid \exists x_1.\ \exists y_1.\ \forall x_2.\ \forall y_2.\ (\Psi_1(x_1, y_2) \wedge \Psi_2(x_2, y_1) \wedge \Psi_3(x_1, y_1)),$$

where $x_1, x_2 \in X$, and $y_1, y_2 \in Y$. The resulting closed strategy formulas are called the *simple one-alternation fragment* of strategy logic. Obviously, the formulas have a single quantifier alternation. We use the abbreviation $(\exists\ \Psi_1,\ \exists\ \Psi_2,\ \Psi_3)$ for simple one-alternation strategy formulas of the form $\exists x_1.\exists y_1.\forall x_2.\forall y_2.\ (\Psi_1(x_1, y_2) \wedge \Psi_2(x_2, y_1) \wedge \Psi_3(x_1, y_1))$.

Notation. For a path formula Ψ and a state s we define the set $\mathsf{Win}_1(s, \Psi) = \{\sigma \in \Sigma \mid \forall \tau \in \Gamma.\ \pi(s, \sigma, \tau) \models \Psi\}$ to denote the set of player-1 strategies that enforce Ψ against all player-2 strategies. We refer to the strategies in $\mathsf{Win}_1(s, \Psi)$ as the *winning* player-1 strategies for Ψ from s. Analogously, we define $\mathsf{Win}_2(s, \Psi) = \{\tau \in \Gamma \mid \forall \sigma \in \Sigma.\ \pi(s, \sigma, \tau) \models \Psi\}$ as the set of winning player-2 strategies for Ψ from s. Using the notation Win_1 and Win_2, the semantics of simple one-alternation strategy formulas can be written as follows: if $\Phi = (\exists\ \Psi_1,\ \exists\ \Psi_2,\ \Psi_3)$, then $[\![\Phi]\!] = \{s \in S \mid \exists \sigma \in \mathsf{Win}_1(s, \Psi_1).\ \exists \tau \in \mathsf{Win}_2(s, \Psi_2).\ \pi(s, \sigma, \tau) \models \Psi_3\}$.

5 Expressive Power of Strategy Logic

In this section we show that ATL^* and several concepts in nonzero-sum games can be expressed in the simple one-alternation fragment of strategy logic. We also show that *game logic*, which was introduced in [2] to express the module-checking problem [17], can be expressed in the one-alternation fragment of strategy logic (but not in the simple one-alternation fragment).

Expressing ATL^* and ATL. For every path formula Ψ, we have

$$\langle\!\langle 1 \rangle\!\rangle(\Psi) = \{s \in S \mid \exists \sigma.\ \forall \tau.\ \pi(s, \sigma, \tau) \models \Psi\} = [\![\exists x.\ \forall y.\ \Psi(x, y)]\!] = [\![(\exists \Psi, \exists \mathbf{tt}, \mathbf{tt})]\!];$$
$$\langle\!\langle 1, 2 \rangle\!\rangle(\Psi) = \{s \in S \mid \exists \sigma.\ \exists \tau.\ \pi(s, \sigma, \tau) \models \Psi\} = [\![\exists x.\ \exists y.\ \Psi(x, y)]\!] = [\![(\exists \mathbf{tt}, \exists \mathbf{tt}, \Psi)]\!].$$

The formulas $\langle\!\langle 2 \rangle\!\rangle(\Psi)$ and $\langle\!\langle \emptyset \rangle\!\rangle(\Psi)$ can be expressed similarly. Hence the logic ATL^* can be defined in the simple one-alternation fragment of strategy logic, and ATL can be defined in the simple one-alternation fragment with unnested path formulas.

Expressing Nash equilibria. In nonzero-sum games the input is a labeled game graph and two path formulas, which express the objectives of the two players. We define Nash equilibria [13] and show that their existence can be expressed in the simple one-alternation fragment of strategy logic.

Payoff profiles. Given a labeled game graph (G, P, L), two path formulas Ψ_1 and Ψ_2, strategies σ and τ for the two players, and a state $s \in S$, the *payoff* for player ℓ, where $\ell \in \{1, 2\}$, is defined as follows:

$$p_\ell(s, \sigma, \tau, \Psi_\ell) = \begin{cases} 1 & \text{if } \pi(s, \sigma, \tau) \models \Psi_\ell; \\ 0 & \text{otherwise.} \end{cases}$$

The *payoff profile* (p_1, p_2) consists of the payoffs $p_1 = p_1(s, \sigma, \tau, \Psi_1)$ and $p_2 = p_2(s, \sigma, \tau, \Psi_2)$ for player 1 and player 2.

Nash equilibria. A *strategy profile* (σ, τ) consists of strategies $\sigma \in \Sigma$ and $\tau \in \Gamma$ for the two players. Given a labeled game graph (G, P, L) and two path formulas Ψ_1 and Ψ_2, the strategy profile (σ^*, τ^*) is a *Nash equilibrium* at a state $s \in S$ if the following two conditions hold:

$$\begin{aligned} &(1)\ \forall \sigma \in \Sigma.\ p_1(s, \sigma, \tau^*, \Psi_1) \le p_1(s, \sigma^*, \tau^*, \Psi_1); \\ &(2)\ \forall \tau \in \Gamma.\ p_2(s, \sigma^*, \tau, \Psi_2) \le p_2(s, \sigma^*, \tau^*, \Psi_2). \end{aligned}$$

The state sets of the corresponding payoff profiles are defined as follows: for $i, j \in \{0, 1\}$, we have

$$\begin{aligned} \mathit{NE}(i, j) = \{s \in S \mid{} & \text{there exists a Nash equilibrium } (\sigma^*, \tau^*) \text{ at } s \text{ such that} \\ & p_1(s, \sigma^*, \tau^*, \Psi_1) = i \text{ and } p_2(s, \sigma^*, \tau^*, \Psi_2) = j\}. \end{aligned}$$

Existence of Nash equilibria. We now define the state sets of the payoff profiles for Nash equilibria by simple one-alternation strategy formulas. The formulas are as follows:

$$\begin{aligned} \mathit{NE}(1,1) &= [\![(\exists \mathbf{tt},\ \exists \mathbf{tt},\ \Psi_1 \wedge \Psi_2)]\!]; \\ \mathit{NE}(0,0) &= [\![(\exists \neg\Psi_2,\ \exists \neg\Psi_1,\ \mathbf{tt})]\!]; \\ \mathit{NE}(1,0) &= \{s \in S \mid \exists \sigma.\ (\exists \tau.\ \pi(s, \sigma, \tau) \models \Psi_1\ \wedge\ \forall \tau'.\ \pi(s, \sigma, \tau') \models \neg\Psi_2)\} \\ &= [\![(\exists \neg\Psi_2,\ \exists \mathbf{tt},\ \Psi_1)]\!]; \\ \mathit{NE}(0,1) &= [\![(\exists \mathbf{tt},\ \exists \neg\Psi_1,\ \Psi_2)]\!]. \end{aligned}$$

Expressing secure equilibria. A notion of conditional competitiveness in nonzero-sum games was formalized by introducing secure equilibria [5]. We show that the existence of secure equilibria can be expressed in the simple one-alternation fragment of strategy logic.

Lexicographic ordering of payoff profiles. We define two lexicographic orderings $\preceq_1$ and $\preceq_2$ on payoff profiles. For two payoff profiles (p_1, p_2) and (p_1', p_2'), we have

$$(p_1, p_2) \preceq_1 (p'_1, p'_2) \text{ iff } (p_1 \le p'_1) \lor (p_1 = p'_1 \land p_2 \ge p'_2);$$
$$(p_1, p_2) \preceq_2 (p'_1, p'_2) \text{ iff } (p_2 \le p'_2) \lor (p_2 = p'_2 \land p_1 \ge p'_1).$$

Secure equilibria. A secure equilibrium is a Nash equilibrium with respect to the lexicographic preference orderings $\preceq_1$ and $\preceq_2$ on payoff profiles for the two players. Formally, given a labeled game graph (G, P, L) and two path formulas Ψ_1 and Ψ_2, a strategy profile (σ^*, τ^*) is a *secure equilibrium* at a state $s \in S$ if the following two conditions hold:

(1) $\forall \sigma \in \Sigma.\ (p_1(s, \sigma, \tau^*, \Psi_1), p_2(s, \sigma, \tau^*, \Psi_2)) \preceq_1 (p_1(s, \sigma^*, \tau^*, \Psi_1), p_2(s, \sigma^*, \tau^*, \Psi_2))$;

(2) $\forall \tau \in \Gamma.\ (p_1(s, \sigma^*, \tau, \Psi_1), p_2(s, \sigma^*, \tau, \Psi_2)) \preceq_2 (p_1(s, \sigma^*, \tau^*, \Psi_1), p_2(s, \sigma^*, \tau^*, \Psi_2))$.

The state sets of the corresponding payoff profiles are defined as follows: for $i, j \in \{0, 1\}$, we have

$$\begin{aligned} SE(i,j) = \{s \in S \mid{}& \text{there exists a secure equilibrium } (\sigma^*, \tau^*) \text{ at } s \text{ such that} \\ & p_1(s, \sigma^*, \tau^*, \Psi_1) = i \text{ and } p_2(s, \sigma^*, \tau^*, \Psi_2) = j\}. \end{aligned}$$

It follows from the definitions that the sets $SE(i,j)$, for $i, j \in \{0,1\}$, can be expressed in the one-alternation fragment (in the $\exists\forall$ fragment). The state sets of maximal payoff profiles for secure equilibria are defined as follows: for $i, j \in \{0, 1\}$, we have

$$MS(i,j) = \{s \in SE(i,j) \mid \text{if } s \in SE(i',j'),\ \text{then } (i',j') \preceq_1 (i,j) \land (i',j') \preceq_2 (i,j)\}.$$

The following alternative characterizations of these sets are established in [5]:

$$\begin{aligned} MS(1,0) &= \{s \in S \mid \mathsf{Win}_1(s, \Psi_1 \land \neg\Psi_2) \neq \emptyset\}; \\ MS(0,1) &= \{s \in S \mid \mathsf{Win}_2(s, \Psi_2 \land \neg\Psi_1) \neq \emptyset\}; \\ MS(1,1) &= \{s \in S \mid \exists \sigma \in \mathsf{Win}_1(s, \Psi_2 \Rightarrow \Psi_1).\ \exists \tau \in \mathsf{Win}_2(s, \Psi_1 \Rightarrow \Psi_2). \\ &\qquad \pi(s, \sigma, \tau) \models \Psi_1 \land \Psi_2\}; \\ MS(0,0) &= S \setminus (MS(1,0) \cup MS(0,1) \cup MS(1,1)). \end{aligned}$$

Existence of secure equilibria. From the alternative characterizations of the state sets of the maximal payoff profiles for secure equilibria, it follows that these sets can be defined by simple one-alternation strategy formulas. The formulas are as follows:

$$\begin{aligned} MS(1,0) &= [\![(\exists(\Psi_1 \land \neg\Psi_2),\ \exists\mathbf{tt},\ \mathbf{tt})]\!]; \\ MS(0,1) &= [\![(\exists\mathbf{tt},\ \exists(\Psi_2 \land \neg\Psi_1),\ \mathbf{tt})]\!]; \\ MS(1,1) &= [\![(\exists(\Psi_2 \Rightarrow \Psi_1),\ \exists(\Psi_1 \Rightarrow \Psi_2),\ \Psi_1 \land \Psi_2)]\!]. \end{aligned}$$

The set $MS(0,0)$ can be obtained by complementing the disjunction of the three formulas for $MS(1,0)$, $MS(0,1)$, and $MS(1,1)$.

Game logic and module checking. The syntax of *game logic* [2] is as follows. State formulas have the form $\exists\{1\}.\ \theta$ or $\exists\{2\}.\ \theta$, where θ is a tree formula.

Tree formulas are (a) state formulas, (b) boolean combinations of tree formulas, and (c) either $\exists\Psi$ or $\forall\Psi$, where Ψ is a path formula. Informally, the formula $\exists\{1\}.\ \theta$ is true at a state if there is a strategy σ for player 1 such that the tree formula θ is satisfied in the tree that is generated by fixing the strategy σ for player 1 (see [2] for details). Game logic can be defined in the one-alternation fragment of strategy logic (but not in the simple one-alternation fragment). The following example illustrates how to translate a state formula of game logic into a one-alternation strategy formula:

$$[\![\exists\{1\}.(\exists\Psi_1 \wedge \forall\Psi_2 \vee \forall\Psi_3)]\!] = [\![\exists x.\ (\exists y_1.\ \Psi_1(x,y_1) \wedge \forall y_2.\ \Psi_2(x,y_2) \vee \forall y_3.\ \Psi_3(x,y_3)]\!]$$

Consequently, the module-checking problem [17] can be expressed by one-alternation strategy formulas.

The following theorem compares the expressive power of strategy logic and its fragments with ATL^*, game logic, the alternating-time μ-calculus [2, 16], and monadic second-order logic [23, 26] (see [6] for proofs).

Theorem 1. *1. The expressiveness of the simple one-alternation fragment of strategy logic coincides with* ATL^**, and the one-alternation fragment of strategy logic is more expressive than* ATL^**.*
2. The one-alternation fragment of strategy logic is more expressive than game logic, and game logic is more expressive than the simple one-alternation fragment of strategy logic.
3. The alternating-time μ*-calculus is not as expressive as the alternation-free fragment of strategy logic, and strategy logic is not as expressive as the alternating-time* μ*-calculus.*
4. Monadic second order logic is more expressive than strategy logic.

6 Model Checking Strategy Logic

In this section we solve the model-checking problem for strategy logic. We encode strategies by using strategy trees. We reason about strategy trees using tree automata, making our solution similar to Rabin's usage of tree automata for solving the satisfiability problem of monadic second-order logic [23]. We give the necessary definitions and proceed with the algorithm.

Strategy trees and tree automata. Given a finite set Υ of directions, an *Υ-tree* is a set $T \subseteq \Upsilon^*$ such that if $x \cdot \upsilon \in T$, where $\upsilon \in \Upsilon$ and $x \in \Upsilon^*$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $\upsilon \in \Upsilon$ and $x \in T$, the node x is the *parent* of $x \cdot \upsilon$. Each node $x \neq \varepsilon$ of T has a *direction* in Υ. The direction of the root is the symbol $\perp$ (we assume that $\perp \notin \Upsilon$). The direction of a node $x \cdot \upsilon$ is υ. We denote by $dir(x)$ the direction of node x. An Υ-tree T is a *full infinite tree* if $T = \Upsilon^*$. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$, and for every $x \in \pi$ there exists a unique $\upsilon \in \Upsilon$ such that $x \cdot \upsilon \in \pi$.

Given two finite sets Υ and Λ, a *Λ-labeled Υ-tree* is a pair $\langle T, \rho\rangle$, where T is an Υ-tree, and ρ: $T \to \Lambda$ maps each node of T to a letter in Λ. When Υ and Λ

are not important or clear from the context, we call $\langle T, \rho \rangle$ a labeled tree. We say that an $((\Upsilon \cup \{\bot\}) \times \Lambda)$-labeled Υ-tree $\langle T, \rho \rangle$ is Υ-*exhaustive* if for every node $z \in T$, we have $\rho(z) \in \{dir(z)\} \times \Lambda$.

Consider a game graph $G = ((S, E), (S_1, S_2))$. For $\alpha \in \{1, 2\}$, a strategy σ: $S^* \cdot S_\alpha \to S$ can be encoded by an S-labeled S-tree $\langle S^*, \rho \rangle$ by setting $\sigma(v) = \rho(v)$ for every $v \in S^* \cdot S_\alpha$. Notice that σ may be encoded by many different trees. Indeed, for a node $v = s_0 \cdots s_n$ such that either $s_n \in S_{3-\alpha}$ or there exists some i such that $(s_i, s_{i+1}) \notin E$, the label $\rho(v)$ may be set arbitrarily. We may encode k different strategies by considering an S^k-labeled S-tree. Given a letter $\lambda \in S^k$, we denote by λ_i the projection of λ on its i-th coordinate. In this case, the i-th strategy is $\sigma_i(v) = \rho(v)_i$ for every $v \in S^* \cdot S_\alpha$. Notice that the different encoded strategies may belong to different players. We refer to such trees as *strategy trees*, and from now on, we may refer to a strategy as a tree $\langle S^*, \sigma \rangle$. In what follows we encode strategies by strategy trees. We construct tree automata that accept the strategy assignments that satisfy a given formula of strategy logic.

We use tree automata to reason about strategy trees. As we only use well-known results about such automata, we do not give a full formal definition, and refer the reader to [25]. Here, we use *alternating parity tree automata* (APTs). The language of an automaton is the set of labeled trees that it accepts. The size of an automaton is measured by the number of states, and the index, which is a measure of the complexity of the acceptance (parity) condition. The important qualities of automata that are needed for this paper are summarized in Theorem 2 below.

Theorem 2. *1. Given an* LTL *formula* Ψ*, we can construct an APT* $\mathcal{A}_\Psi$ *with* $2^{O(|\Psi|)}$ *states and index* 3 *such that* $\mathcal{A}_\Psi$ *accepts all labeled trees all of whose paths satisfy* Ψ [27].

2. *Given two APTs* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *with* n_1 *and* n_2 *states and indices* k_1 *and* k_2*, respectively, we can construct APTs for the conjunction and disjunction of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *with* $n_1 + n_2$ *states and index* $\max(k_1, k_2)$*. We can also construct an APT for the complementary language of* $\mathcal{A}_1$ *with* n_1 *states and index* k_1 [20].
3. *Given an APT* $\mathcal{A}$ *with* n *states and index* k *over the alphabet* $\Lambda \times \Lambda'$*, we can construct an APT* $\mathcal{A}'$ *that accepts a labeled tree over the alphabet* Λ *if some extension (or all extensions) of the labeling with labels from* Λ' *is accepted by* $\mathcal{A}$*. The number of states of* $\mathcal{A}'$ *is exponential in* $n \cdot k$*, and its index is linear in* $n \cdot k$ [20].
4. *Given an APT* $\mathcal{A}$ *with* n *states and index* k*, we can check whether the language of* $\mathcal{A}$ *is empty or universal in time exponential in* $n \cdot k$ [9, 20].

Model-checking algorithm. The complexity of the model-checking algorithm for strategy formulas depends on the number of quantifier alternations of a formula. We now formally define the alternation depth of a closed strategy formula. The *alternation depth of a variable* of a closed strategy formula is the number of quantifier switches ($\exists\forall$ or $\forall\exists$) that bind the variable. The *alternation depth of a closed strategy formula* is the maximal alternation depth of a variable occurring in the formula.

Given a strategy formula Φ, we construct by induction on the structure of the formula a nondeterministic parity tree (NPT) automaton that accepts the set of strategy assignments that satisfy the formula. Without loss of generality, we assume that the variables in $X \cup Y$ are not reused; that is, in a closed strategy formula, there is a one-to-one and onto relation between the variables and the quantifiers.

Theorem 3. *Given a labeled game graph $\mathcal{G}$ and a closed strategy formula Φ of alternation depth d, we can compute the set $[\![\Phi]\!]$ of states in time proportional to d-EXPTIME in the size of $\mathcal{G}$, and $(d+1)$-EXPTIME in the size of Φ. If Φ contains only unnested path formulas, then the complexity in the size of the formula reduces to d-EXPTIME.*

Proof. The case where closed strategy formula Φ is used as a state formula in a larger formula Φ', is solved by first computing the set of states satisfying Φ, adding this information to the labeled game graph $\mathcal{G}$, and then computing the set of states satisfying Φ'. In addition, if d is the alternation-depth of Φ then Φ is a boolean combination of closed strategy formulas of alternation depth at most d. Thus, it suffices to handle a closed strategy formula, and reduce the boolean reasoning to intersection, union, and complementation of the respective sets.

Consider a strategy formula Φ. Let $Z = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ be the set of variables used in Φ. Consider the alphabet S^{n+m} and an S^{n+m}-labeled S-tree σ. For a variable $v \in X \cup Y$, we denote by σ_v the strategy that stands in the location of variable v and for a set $Z' \subseteq Z$ we denote by $\sigma_{Z'}$ the set of strategies for the variables in Z'. We now describe how to construct an APT that accepts the set of strategy assignments that satisfy Φ. We build the APT by induction on the structure of the formula. For a subformula Φ' we consider the following cases.

Case 1. $\Phi' = \Psi(x, y)$ —by Theorem 2 we can construct an APT $\mathcal{A}$ that accepts trees all of whose paths satisfy Ψ. According to Theorem 2, $\mathcal{A}$ has $2^{O(|\Psi|)}$ states.

Case 2. $\Phi' = \Phi_1 \wedge \Phi_2$ —given APTs $\mathcal{A}_1$ and $\mathcal{A}_2$ that accept the set of strategy assignments that satisfy Φ_1 and Φ_2, respectively; we construct an APT $\mathcal{A}$ for the conjunction of $\mathcal{A}_1$ and $\mathcal{A}_2$. According to Theorem 2, $|\mathcal{A}| = |\mathcal{A}_1| + |\mathcal{A}_2|$ and the index of $\mathcal{A}$ is the maximum of the indices of $\mathcal{A}_1$ and $\mathcal{A}_2$.

Case 3. $\Phi' = \exists x.\Phi_1$ —given an APT $\mathcal{A}_1$ that accepts the set of strategy assignments that satisfy Φ_1 we do the following. According to Theorem 2, we can construct an APT $\mathcal{A}'$ that accepts a tree iff there exists a way to extend the labeling of the tree with a labeling for the strategy for x such that the extended tree is accepted by $\mathcal{A}_1$. The number of states of $\mathcal{A}'$ is exponential in $n \cdot k$ and its index is linear in $n \cdot k$. The cases where $\Phi' = \exists y.\Phi_1$, $\Phi' = \forall x.\Phi_1$, and $\Phi' = \forall y.\Phi_1$ are handled similarly.

We note that for a closed strategy formula Φ, the resulting automaton reads $S^{\emptyset}$-labeled S-trees. Thus, the input alphabet of the automaton has a single input letter and it only reads the structure of the S-tree.

The above construction starts with an automaton that is exponential in the size of a given LTL formula and incurs an additional exponent for every quantifier.

In order to pay an exponent 'only' for every quantifier alternation, we have to use nondeterministic and universal automata, and maintain them in this form as long as possible. Nondeterministic automata are good for existential quantification, which comes to them for free, and universal automata are good for universal quantification. By careful analysis of the quantifier alternation hierarchy, we can choose to create automata of the right kind (nondeterministic or universal), and maintain them in this form under disjunctions and conjunctions. Then, the complexity is $d + 1$ exponents in the size of the formula and d exponents in the size of the game.

Consider the case where only unnested path formulas are used. Then, given a path formula $\Psi(x, y)$, we construct an APT $\mathcal{A}$ that accepts trees all of whose paths satisfy Ψ. As $\Psi(x, y)$ does not use nesting of temporal operators, we can construct $\mathcal{A}$ with a linear number of states in the size of Ψ.[1] It follows that the total complexity is d exponents in the size of the formula and d exponents in the size of the game. Thus in the case of unnested path formulas one exponent can be removed. The exact details are omitted due to lack of space. ∎

One-alternation fragment. Since ATL* can be expressed in the simple one-alternation fragment of strategy logic, it follows that model checking simple one-alternation strategy formulas is 2EXPTIME-hard [2]. Also, since module checking can be expressed in the one-alternation fragment, it follows that model checking one-alternation strategy formulas with unnested path formulas is EXPTIME-hard [17]. These lower bounds together with Theorem 3 yield the following results.

Theorem 4. *Given a labeled game graph $\mathcal{G}$ and a closed one-alternation strategy formula Φ, the computation of $[\![\Phi]\!]$ is EXPTIME-complete in the size of $\mathcal{G}$, and 2EXPTIME-complete in the size of Φ. If Φ contains only unnested path formulas, then the complexity in the size of the formula is EXPTIME-complete.*

Model checking the simple one-alternation fragment. We now present a model-checking algorithm for the simple one-alternation fragment of strategy logic, with better complexity than the general algorithm. We first present a few notations.

Notation. For a labeled game graph $\mathcal{G}$ and a set $U \subseteq S$ of states, we denote by $\mathcal{G} \upharpoonright U$ the restriction of the labeled game graph to the set U, and we use the notation only when for all states $u \in U$, we have $E(u) \cap U \neq \emptyset$; i.e., all states in U have a successor in U. A path formula Ψ is *infinitary* if the set of paths that satisfy Ψ is independent of all finite prefixes. The classical Büchi, coBüchi, parity, Rabin, Streett, and Müller conditions are all infinitary conditions. Every LTL objective on a labeled game graph can be reduced to an infinitary condition, such as a parity or Müller condition, on a modified game graph.

[1] For a single temporal operator the number of states is constant, and boolean combinations between two automata may lead to an automaton whose size is the product of the sizes of the two automata. The number of multiplications is at most logarithmic in the size of the formula, resulting in a linear total number of states.

Lemma 1. *Let $\mathcal{G}$ be a labeled game graph, and let $\Phi = (\exists\ \Psi_1,\ \exists\ \Psi_2,\ \Psi_3)$ be a simple one-alternation strategy formula with path formulas Ψ_1, Ψ_2, and Ψ_3 such that Ψ_1 and Ψ_2 are infinitary. Let $W_1 = \langle\langle 1\rangle\rangle(\Psi_1)$ and $W_2 = \langle\langle 2\rangle\rangle(\Psi_2)$. Then $[\![\Phi]\!] = \langle\langle 1,2\rangle\rangle(\Psi_1 \wedge \Psi_2 \wedge \Psi_3)$ in the restricted graph $\mathcal{G} \upharpoonright (W_1 \cap W_2)$.*

Lemma 2. *Let $\mathcal{G}$ be a labeled game graph, and let $\Phi = (\exists\ \Psi_1,\ \exists\ \Psi_2,\ \Psi_3)$ be a simple one-alternation strategy formula with unnested path formulas Ψ_1, Ψ_2, and Ψ_3. Let $W_1 = \langle\langle 1\rangle\rangle(\Psi_1)$ and $W_2 = \langle\langle 2\rangle\rangle(\Psi_2)$. Then $[\![\Phi]\!] = \langle\langle 1,2\rangle\rangle(\Psi_1 \wedge \Psi_2 \wedge \Psi_3) \cap W_1 \cap W_2$.*

Theorem 5. *Let $\mathcal{G}$ be a labeled game graph with n states, and let $\Phi = (\exists\ \Psi_1,\ \exists\ \Psi_2,\ \Psi_3)$ be a simple one-alternation strategy formula.*

1. *We can compute the set $[\![\Phi]\!]$ of states in $n^{2^{O(|\Phi|)}} \cdot 2^{2^{O(|\Phi|\cdot\log|\Phi|)}}$ time; hence for formulas Φ of constant length the computation of $[\![\Phi]\!]$ is polynomial in the size of $\mathcal{G}$. The computation of $[\![\Phi]\!]$ is 2EXPTIME-complete in the size of Φ.*
2. *If Ψ_1, Ψ_2, and Ψ_3 are unnested path formulas, then there is a ATL^* formula Φ' with unnested path formulas such that $|\Phi'| = O(|\Psi_1| + |\Psi_2| + |\Psi_3|)$ and $[\![\Phi]\!] = [\![\Phi']\!]$. Therefore $[\![\Phi]\!]$ can be computed in polynomial time.*

Theorem 5 follows from Lemmas 1 and 2 (see [6] for the proofs). We present some details only for part (1): given Ψ_1, Ψ_2, and Ψ_3 as parity conditions, from Lemma 1, it follows that $[\![(\exists\Psi_1,\ \exists\Psi_2, \Psi_3)]\!]$ can be computed by first solving two parity games, and then model checking a graph with a conjunction of parity conditions (i.e., a Streett condition). Since an LTL formula Ψ can be converted to an equivalent deterministic parity automaton with $2^{2^{O(|\Psi|\cdot\log|\Psi|)}}$ states and $2^{O(|\Psi|)}$ parities (by converting Ψ to a nondeterministic Büchi automaton, and then determinizing), applying an algorithm for solving parity games [14] and a polynomial-time algorithm for model checking Streett conditions, we obtain the desired upper bound. Observe that the model-checking complexity of the simple one-alternation fragment of strategy logic with unnested path formulas, as well as the program complexity of the simple one-alternation fragment (i.e., the complexity in terms of the game graph, for formulas of bounded size), are exponentially better than the corresponding complexities of the full one-alternation fragment.

Acknowledgments. This research was supported in part by the Swiss National Science Foundation and by the NSF grants CCR-0225610 and CCR-0234690.

References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *16th ICALP*, LNCS 372, pages 1–17. Springer, 1989.
2. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002.
3. A. Blass, Y. Gurevich, L. Nachmanson, and M. Veanes. Play to test. In *5th FATES*, LNCS 3997, pages 32–46. Springer, 2005.

4. K. Chatterjee and T.A. Henzinger. Assume guarantee synthesis. In *30th TACAS*, LNCS 4424, pages 261–275. Springer, 2007.
5. K. Chatterjee, T.A. Henzinger, and M. Jurdziński. Games with secure equilibria. In *19th LICS*, pages 160–169. IEEE Computer Society Press, 2004.
6. K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy logic. Technical Report UCB/EECS-2007-78, UC Berkeley, 2007.
7. L. de Alfaro and T.A. Henzinger. Interface automata. In *9th FASE*, pages 109–120. ACM press, 2001.
8. D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989.
9. E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *5th CAV*, LNCS 697, pages 385–396. Springer, 1993.
10. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *7th POPL*, pages 163–173. ACM Press, 1980.
11. T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. *Information and Computation*, 173(1):64–81, 2002.
12. L. Kaiser. Game quantification on automatic structures and hierarchical model checking games. In *15th CSL*, LNCS 4207, pages 411–425. Springer, 2006.
13. J.F. Nash Jr. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences USA*, 36:48–49, 1950.
14. M. Jurdziński. Small progress measures for solving parity games. In *17th STACS*, LNCS 1770, pages 290–301. Springer, 2000.
15. J.A.W. Kamp. *Tense Logic and the Theory of Order*. PhD thesis, UCLA, 1968.
16. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
17. O. Kupferman, M.Y. Vardi, and P. Wolper. Module checking. *Information and Computation*, 164:322–344, 2001.
18. D.A. Martin. Borel determinacy. *Annals of Mathematics*, 65:363–371, 1975.
19. R. Milner. An algebraic definition of simulation between programs. In *2nd IJCAI*, pages 481–489. British Computer Society, 1971.
20. D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.
21. G. Owen. *Game Theory*. Academic Press, 1995.
22. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *16th POPL*, pages 179–190. ACM Press, 1989.
23. M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.
24. P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *IEEE Transactions on Control Theory*, 77:81–98, 1989.
25. W. Thomas. On the synthesis of strategies in infinite games. In *12th STACS*, LNCS 900, pages 1–13. Springer, 1995.
26. W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, Beyond Words, chapter 7, pages 389–455. Springer, 1997.
27. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.