Automated Synthesis of Generalized Invariant Strategies via Counterexample-Guided Strategy Refinement

Kailun Luo,[1,2] **Yongmei Liu**[2*]

[1] School of Cyberspace Security, Dongguan University of Technology, Dongguan 523808, China
[2] Dept. of Computer Science, Sun Yat-sen University, Guangzhou 510006, China
luokl@dgut.edu.cn, ymliu@mail.sysu.edu.cn

Abstract

Strategy synthesis for multi-agent systems has proved to be a hard task, even when limited to two-player games with safety objectives. Generalized strategy synthesis, an extension of generalized planning which aims to produce a single solution for multiple (possibly infinitely many) planning instances, is a promising direction to deal with the state-space explosion problem. In this paper, we formalize the problem of generalized strategy synthesis in the situation calculus. The synthesis task involves second-order theorem proving generally. Thus we consider strategies aiming to maintain invariants; such strategies can be verified with first-order theorem proving. We propose a sound but incomplete approach to synthesize invariant strategies by adapting the framework of counterexample-guided refinement. The key idea for refinement is to generate a strategy using a model checker for a game constructed from the counterexample, and use it to refine the candidate general strategy. We implemented our method and did experiments with a number of game problems. Our system can successfully synthesize solutions for most of the domains within a reasonable amount of time.

Introduction

Two player games with reachability or safety objectives require a player to reach or avoid a designated set of target configurations. They are closely related to the synthesis of reactive systems [Pnueli and Rosner 1989], and logical tasks such as first-order model checking [Gradel et al. 2007].

While multi-agent systems have received much attention, verifying strategic abilities and synthesizing strategies for them have proved to be hard tasks. In Strategy Logic, the model checking problem is non-elementarily decidable [Chatterjee, Henzinger, and Piterman 2010]. In Alternating-time Temporal Logic, the model checking problem is P-complete wrt the state space, even considering *memoryless strategies* [Alur, Henzinger, and Kupferman 2002; Bulling, Dix, and Jamroga 2010]; it encounters the state-space explosion problem with the increasing number of propositions. For games with reachability or safety objectives, the problem of computing *winning regions* for winning strategies, is P-complete [Greenlaw, Hoover, and Ruzzo 1995]; the computation approaches suffer from performance issues [Eén et al. 2015].

For domains with similar structures, generalized strategy synthesis is a promising direction to deal with the state-space explosion problem. Generalized strategy synthesis can be viewed as a multi-agent extension of generalized planning, which aims to generate one solution that works for multiple (possibly infinitely many) planning problems [Srivastava, Immerman, and Zilberstein 2011; Hu and De Giacomo 2011]. Different from the objective to design on-line systems to play arbitrary games in general game playing [Genesereth, Love, and Pell 2005], generalized strategy synthesis focuses on off-line strategy synthesis for multiple game instances sharing similar structures. If a generalized solution is generated, for any game instance with similar structures, we could efficiently produce a concrete solution from the generalized one.

A typical example is the n-Nim game, played with n heaps of pebbles. Two players take turns removing pebbles from a heap. On each turn, a player must remove at least one pebble, and may remove any number of pebbles provided that they all come from the same heap. The player to remove the last pebble wins. The safety objective is to win the game whenever it ends. The task is to synthesize a strategy that works for all instances of the n-Nim game.

Wu et al. [2020] consider the automated synthesis of generalized winning strategies for impartial combinatorial games. They first generate a winning formula using given templates by trying to include or exclude states corresponding to counterexamples generated during the verification process. They then partition the winning formula into several sub-formulas, and attempt to find an action to take under the condition of each sub-formula via heuristic methods. While obtaining good performance, their framework has some limitations. It only considers numerical variables and the linear relations among them; thus domains and strategies which involve more expressive formalizations can not be handled. It requires providing state constraints to denote legal states, which is not an easy task for some domains.

In this paper, we consider a more general form of synthesis, namely generalized strategy synthesis for finite-state turn-based two-player games with perfect information and safety objectives. We formalize the problem of generalized strategy synthesis as a theorem-proving task in the situation

*Corresponding author

calculus. We use a basic action theory to represent a class of game instances with similar structures. We focus on postdiction strategies of the form $\varphi?; \pi a.a; \psi?$, where φ and ψ are first-order formulas, which intuitively means: when φ holds, do any action to make ψ true.

We propose an automated method, sound but incomplete, to synthesize postdiction strategies called invariant strategies, by adapting the idea of counterexample-guided inductive synthesis [Solar-Lezama et al. 2006] in Formal Methods. Intuitively, a postdiction strategy $\varphi?; \pi a.a; \psi?$ for a player p is an invariant strategy if we have: whenever it's p's turn to move and φ holds, p can execute an action to enforce ψ; whenever it's the opponent's turn to move and ψ holds, any action the opponent can execute makes φ true. Invariant strategies can be verified with first-order theorem proving. To synthesize an invariant strategy, we first propose a rough postdiction strategy and verify if it is an invariant strategy; if not, we use a counterexample to refine the strategy and continue the process. The counterexample is used to induce a finite set of game instances, each of which is solved via the existing model-checking methods to obtain a $winning\ strategy$. Depending on whether the counterexample is $reachable$ via a winning strategy, we refine the postdiction strategy accordingly.

We implemented our method and experimented with a number of combinatorial games, grid games and a game variant of a protocol for leader election. Our system can successfully synthesize solutions for most of the domains within a reasonable amount of time.

Preliminaries

The situation calculus is a many-sorted first-order (FO) language with limited second-order features for representing dynamically changing worlds [McCarthy and Hayes 1969; Reiter 2001]. In a situation calculus language, there are three disjoint sorts: $action$ for actions, $situation$ for situations and $object$ for everything else. Constant S_0 denotes the only initial situation. Binary function $do(a, s)$ represents the situation resulting from performing action a in situation s. Binary relation $Poss(a, s)$ denotes that action a is executable in situation s. Binary relation $s \sqsubseteq s'$ means that s' can be obtained via a sequence of actions from s. Notation $exec(s)$ means that situation s is reachable from the initial situation S_0 via a sequence of executable actions. Dynamic properties are captured by predicates and functions called $fluents$ whose values vary from situation to situation. If a formula only refers to a particular situation τ, we call it uniform in τ. Let φ be a uniform formula. We use $\varphi_{\downarrow}$ to denote φ with all situation arguments removed, and we call it a $situation\text{-}suppressed\ formula$. For a situation-suppressed formula ϕ, we use $\phi[s]$ to denote the formula obtained from ϕ by restoring s as the situation arguments to all fluents.

A dynamic domain is specified by a basic action theory (BAT) of the form:

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}, \text{where}$$

1. Σ is the set of the foundational axioms for situations;
2. $\mathcal{D}_{una}$ is a set of unique name axioms for actions;
3. $\mathcal{D}_{ap}$ is a set of precondition axioms (PAs) of the form $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$, where $\Pi_A(\vec{x}, s)$ is uniform in s, denoting the executability condition for action $A(\vec{x})$;
4. $\mathcal{D}_{ss}$ is a set of successor state axioms (SSAs) of the form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$ for relational fluents and $f(\vec{x}, do(a, s)) = y \equiv \Phi_f(\vec{x}, y, a, s)$ for functional fluents, where $\Phi_F(\vec{x}, a, s)$ and $\Phi_f(\vec{x}, y, a, s)$ are uniform in s, denoting how the values of fluents will be changed or maintained via actions;
5. $\mathcal{D}_{S_0}$, the initial database, is a set of sentences uniform in S_0.

The PAs and SSAs for the 2-Nim game are presented:

Example 1. Let fluent $n(s)$ (resp. $m(s)$) denote the number of pebbles in the first (resp. second) heap; let action $rn(x)$ (resp. $rm(x)$) represent that a player removes x pebbles from the first (resp. second) heap. Then the PAs and SSAs are as follows:

- $Poss(rn(x), s) \equiv n(s) \geq x \wedge x > 0$
- $Poss(rm(x), s) \equiv m(s) \geq x \wedge x > 0$
- $n(do(a, s)) = y \equiv \exists x.a = rn(x) \wedge n(s) = x + y \vee n(s) = y \wedge \exists x.a = rm(x)$
- $m(do(a, s)) = y \equiv \exists x.a = rm(x) \wedge m(s) = x + y \vee m(s) = y \wedge \exists x.a = rn(x)$

To represent and reason about complex actions, Levesque *et al.* [1997] introduced a high-level programming language called Golog, whose syntax is as follows:

$$\delta ::= \alpha \mid \varphi? \mid (\delta_1; \delta_2) \mid (\delta_1 | \delta_2) \mid \pi x.\delta \mid \delta^*,$$

where α is an action term; $\varphi?$ is an action, testing whether a situation-suppressed formula φ holds; program $\delta_1; \delta_2$ represents the sequential execution of δ_1 and δ_2; program $\delta_1 | \delta_2$ denotes the non-deterministic choice between δ_1 and δ_2; program $\pi x.\delta$ denotes the non-deterministic choice of a value for parameter x in δ; program δ^* means executing program δ for a non-deterministic number of times. The semantics of Golog is specified by $Do(\delta, s, s')$, which macro-expands into a situation calculus formula, saying that it is possible to reach situation s' from situation s by executing a sequence of actions specified by program δ.

Formally, $Do(\delta, s, s')$ is inductively defined as follows:

1. $Do(\alpha, s, s') \doteq Poss(\alpha, s) \wedge s' = do(\alpha, s)$.
2. $Do(\phi?, s, s') \doteq \phi[s] \wedge s = s'$.
3. $Do(\delta_1; \delta_2, s, s') \doteq \exists s''.Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$.
4. $Do(\delta_1 | \delta_2, s, s') \doteq Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$.
5. $Do((\pi x)\delta(x), s, s') \doteq (\exists x)Do(\delta(x), s, s')$.
6. $Do(\delta^*, s, s') \doteq (\forall P).\{(\forall s_1)P(s_1, s_1) \wedge (\forall s_1, s_2, s_3) [P(s_1, s_2) \wedge Do(\delta, s_2, s_3) \supset P(s_1, s_3)]\} \supset P(s, s')$.

Besides the $evaluation\ semantics$ above, Giacomo *et al.* [2000] defined the $transition\ semantics$. In transition semantics, a configuration is a pair (δ, s), where δ is a program remaining to be executed and s is a situation. A special program nil represents the fact that nothing remains to be performed. Predicate $Trans(\delta, s, \delta, s')$ intuitively means a transition from configuration (δ, s) to (δ', s') in one step. Predicate $Final(\delta, s)$ denotes whether program δ can legally terminate in situation s. Their formal definitions can

be referred in [Giacomo, Lespérance, and Levesque 2000]. With $Trans$ and $Final$, Do can be defined as follows:

$Do(\delta, s, s') \equiv \exists \delta'.Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$,

where $Trans^*$ denotes the reflexive transitive closure of $Trans$.

Regression is an important computational mechanism for reasoning about actions, to reduce the evaluation of a sentence to a FO theorem-proving task in the initial database. For our purpose, we present the *one-step* regression here.

Definition 1. Given a BAT $\mathcal{D}$, we use $\mathcal{R}[\varphi]$ to denote the formula obtained from φ by the following steps:

1. For each functional fluent term $f(\vec{t}, do(\alpha, \sigma))$, replace the current formula ψ with $(\exists y).\Phi_f(\vec{t}, y, \alpha, \sigma) \wedge \psi[f(\vec{t}, do(\alpha, \sigma))/y]$, where $\psi[t/y]$ denotes the result of replacing all occurrences of t in ψ by y;
2. Replace each fluent atom $F(\vec{t}, do(\alpha, \sigma))$ with $\Phi_F(\vec{t}, \alpha, \sigma)$;
3. Replace each atom $Poss(\alpha, \sigma)$ with $\Pi_\alpha(\sigma)$;
4. Further simplify the result by using $\mathcal{D}_{una}$.

Example 2. If φ is $n(do(rn(3), s)) = 5$, then $\mathcal{R}[\varphi]$ is $\exists y.n(s) = 3 + y \wedge y = 5$, which is equivalent to $n(s) = 8$.

Proposition 1. $\mathcal{D} \models \varphi \equiv \mathcal{R}[\varphi]$.

In this paper, we consider finite-state turn-based two-player games. We assume that players have perfect information, *i.e.*, players know all the information of the state. We call them FTTP games for short. Formally,

Definition 2. A FTTP game G is a tuple $(Q_1, Q_2, q_0, \mathcal{A}_1, \mathcal{A}_2, \delta_1, \delta_2, \Delta, l)$, where, for $i = 1, 2$,

- Q_i is a finite set of states for player i;
- $q_0 \in Q_1$ is the starting state;
- $\mathcal{A}_i$ is a set of actions for player i;
- $\delta_i : Q_i \times \mathcal{A}_i \to Q_{3-i}$ is a transition function;
- Δ is a set of atomic propositions;
- $l : Q_1 \cup Q_2 \to 2^\Delta$ is a labelling function.

A $play$ for game G is an infinite sequence of states, starting from q_0 and following the transition functions. A $history$ is a finite prefix of a play. We denote $\mathcal{H}$ as the set of all histories of a game.

A $strategy$ $\sigma_i : \mathcal{H} \to \mathcal{A}_i$ for player i ($i = 1, 2$) is a function that maps a history to an action. A strategy is $memoryless$ if the selection of actions only depends on the last state of any history. For games with safety objectives, a player has a strategy to win iff she has a memoryless strategy to do so [Apt and Grädel 2011].

A memoryless strategy σ_i induces a structure called a $play\ region$ which represents all the states resulting from all the possible plays, when player i adopts the strategy. Formally, $G|_{\sigma_i} = (Q_1', Q_2')$, where Q_1' and Q_2' are defined inductively:

1. $q_0 \in Q_1'$;
2. If $q \in Q_i'$, $\sigma_i(q) = a$, and $\delta_i(q, a) = q'$, then $q' \in Q_{3-i}'$;
3. If $q \in Q_{3-i}'$ and $\delta_{3-i}(q, a) = q'$ for some a, then $q' \in Q_i'$.

Given safety condition ϕ for player i, we say that a strategy σ_i is a *safe strategy* if any state in the induced player region $G|_{\sigma_i}$ satisfies ϕ. We call such $G|_{\sigma_i}$ a *safe region*.

Our Representation Framework

In this section, we introduce our framework to represent the problem of synthesizing a postdiction strategy for a finite-state safety game BAT, which represents a set of (possibly infinitely many) FTTP games with safety goals.

We first show how to extend the situation calculus and a basic action theory $\mathcal{D}$ to define a finite-state game BAT that represents a set of (possibly infinitely many) FTTP games. As in [Luo and Liu 2019], we add a new sort $player$ and two constants P_1 and P_2; we add a new fluent $turn(p, s)$ to specify the turn for players in situation s; we add the following axioms:

- $\forall p.(p = P_1 \vee p = P_2) \wedge (P_1 \neq P_2)$, saying that there are only two players;
- $turn(P_1, S_0) \wedge \neg turn(P_2, S_0)$;
- $turn(p, do(a, s)) \equiv \neg turn(p, s)$, saying that two players move alternately.

To represent domains involving arithmetic, we introduce a new sort nat for natural numbers, and we add to $\mathcal{D}$ the second-order axiomatization of Peano arithmetic. Without loss of generalization and to simplify the presentation, we assume all objects are of sort nat, since we can use natural numbers as IDs for objects.

We introduce a finite-state axiom for a game BAT. Let $\vec{x}$ be a tuple of variables. We use $\vec{x} > k$ to denote the formula $x_1 > k \vee \ldots \vee x_i > k$.

Definition 3. Given a game BAT $\mathcal{D}$, we define the finite-state axiom $\mathcal{D}_{fs}$ as the formula $\exists k \forall s.exec(s) \supset B(k, s)$, where $B(k, s)$ is the conjunction of the following formulas:

1. $\forall \vec{x}.\vec{x} > k \supset \neg F(\vec{x}, s)$, for each relational fluent F;
2. $\forall \vec{x}.\vec{x} > k \supset f(\vec{x}, s) = 0$, for each non-unary functional fluent f;
3. $\forall \vec{x}.f(\vec{x}, s) \leq k$, for each functional fluent f;
4. $\forall \vec{x}.\vec{x} > k \supset \neg Poss(A(\vec{x}), s)$, for each action A.

The finite-state axiom means that there is a number k such that all larger numbers are inactive throughout the game. We call k the $game\ active\ number$. Thus each model of the finite-state axiom corresponds to a model with finite domain, hence the model represents a finite-state game.

Definition 4. A game BAT $\mathcal{D}$ is finite-state if $\mathcal{D} \models \mathcal{D}_{fs}$.

We then define our generalized game problems.

Definition 5. A generalized finite-state safety (GFSS) game problem is a tuple $P = \langle \mathcal{D}, p, \phi \rangle$, where $\mathcal{D}$ is a finite-state game BAT, p is a player, and ϕ is a situation-suppressed formula denoting a safety goal for player p to enforce.

A general solution for a set of FFTP games is not always first-order definable, as winning regions for safety games are not always first-order definable [Apt and Grädel 2011]. In this paper, we focus on a special kind of strategies which we call postdiction strategies: such a strategy does any action to enforce a first-order definable condition; it has a simple structure and is closely related to memoryless strategies.

As in [Luo and Liu 2019], we use $\pi a.a$ to mean doing any executable action. Its formal semantics is defined as follows:

$$Do(\pi a.a, s, s') \doteq \exists a.Poss(a, s) \wedge s' = do(a, s).$$

Definition 6. A postdiction strategy is a Golog program of the form $\varphi?; \pi a.a; \psi?$.

Intuitively, the strategy means whenever φ holds, perform any action to make ψ hold.

Example 3. For the 2-Nim game, $n\%2 = 0?; \pi a.a; n\%2 = 1?$ is a postdiction strategy, which means that whenever the number of pebbles in the first heap is even, take any action to make it odd.

We show the relation between memoryless strategies and postdiction strategies. In fact, a postdiction strategy provides a compacted representation of memoryless strategies.

Proposition 2. *Given a FTTP game with a safety objective, a player has a memoryless strategy to win iff she has a postdiction strategy to do so.*

Proof. Given a memoryless strategy ensuring a safe region (Q_1', Q_2'), one can obtain a postdiction strategy $\bigvee_{q\in Q_1'} \bigwedge l(q)?; \pi a.a; \bigvee_{q\in Q_2'} \bigwedge l(q)?$ whose play region is also (Q_1', Q_2'), which make it safe. Given a postdiction strategy $\varphi?; \pi a.a; \psi?$ being a safe strategy, one can obtain a safe memoryless strategy σ as follows. Let $\sigma(h) = a$, if the last state s of history h satisfies φ, where a is such an action that via action a the next state of state s satisfies ψ. □

Note that we can convert a postdiction strategy to an equivalent but more specific one by eliminating the use of $\pi a.a$. The idea is to consider each possible action $A_i(\vec{c})$, $i = 1, \ldots, N$, and perform regression on ψ over $A_i(\vec{c})$.

Definition 7. Given a postdiction strategy $\varphi?; \pi a.a; \psi?$, we define its normalization as the non-deterministic composition of the following programs: for $i = 1, \ldots, N$,

$$\varphi?; \pi\vec{x}.\mathcal{R}[\psi(do(A_i(\vec{x}), s))]_{\downarrow}?; A_i(\vec{x}).$$

Example 4. $n\%2 = 0?; \pi a.a; n\%2 = 1?$ is normalized as

$$n\%2 = 0?; \pi x.(n + x)\%2 = 1?; rn(x).$$

We now define the concept of solutions to GFSS game problems. Given a postdiction strategy $\mathcal{S}$ for player p, we use $\delta_{\mathcal{S}}$ to denote the composite strategy where the opponent adopts the *null strategy*, *i.e.*, $\delta_{\mathcal{S}}$ is defined as $turn(p)?; \mathcal{S} \mid \neg turn(p)?; \pi a.a$. Then we have:

Definition 8. Given a GFSS game problem $P = \langle \mathcal{D}, p, \phi \rangle$ and a postdiction strategy $\mathcal{S}$ for player p, we say that $\mathcal{S}$ is a solution to P or $\mathcal{S}$ is a safe strategy for P if

$$\mathcal{D} \models \forall s \forall \delta. Trans^*(\delta_{\mathcal{S}}^*, S_0, \delta, s) \supset \phi[s] \wedge \exists s'. Trans(\delta_{\mathcal{S}}, s, nil, s').$$

Intuitively, $\mathcal{S}$ is a solution to P if when p adopts $\mathcal{S}$ and the opponent adopts the null strategy to play, any reachable situation satisfies ϕ and at any reachable situation, the strategies always prescribe an action to take. Although we focus on postdiction strategies whose conditions are first-order definable, the *state representation* of the resulting reachable situations is not always first-order definable. In general, to test whether $\mathcal{S}$ is a solution is a second-order theorem-proving task, which is highly undecidable.

Automated Synthesis of Invariant Strategies

This section presents a sound but incomplete method to synthesize safe postdiction strategies called invariant strategies.

The definition of safe postdiction strategies involves second-order theorem proving. In this paper, we focus on safe postdiction strategies that can be verified with first-order theorem proving. We call them invariant strategies.

Intuitively, an invariant strategy enables a player to maintain a first-order definable property that ensures the safety condition, no matter how the opponent acts throughout the games. The verification of an invariant strategy relaxes the requirement of reachability that is needed for a safe postdiction strategy, and thus possibly examines unreachable states of the games.

We say that a postdiction strategy $\varphi?; \pi a.a; \psi?$ for player p is an invariant strategy if the following hold:

1. whenever it's p's turn to move and φ holds, p can execute an action to enforce ψ;
2. whenever it's the opponent's turn to move and ψ holds, any action the opponent can execute makes φ true;
3. both φ and ψ imply the safety condition;
4. any initial state satisfies φ (resp. ψ) if p is P_1 (resp. P_2).

The formal definition is as follows:

Definition 9. Given a GFSS game problem $\langle \mathcal{D}, p, \phi \rangle$, we say that a postdiction strategy $\varphi?; \pi a.a; \psi?$ for player p is an invariant strategy if the following conditions hold:

I1 $\varphi \wedge turn(p) \models \bigvee_{i=1}^{N} \exists \vec{x}.\mathcal{R}[Poss(A_i(\vec{x}), s) \wedge \psi(do(A_i(\vec{x}), s))]_{\downarrow}$;

I2 $\psi \wedge \neg turn(p) \models \bigwedge_{i=1}^{N} \forall \vec{x}.\mathcal{R}[Poss(A_i(\vec{x}), s) \supset \varphi(do(A_i(\vec{x}), s))]_{\downarrow}$;

I3 $\varphi \models \phi$, and $\psi \models \phi$;

I4 if $p = P_1$, $\mathcal{D}_{S_0\downarrow} \models \varphi$; otherwise $\mathcal{D}_{S_0\downarrow} \models \psi$.

Example 5. $n\%2 = 0?; \pi a.a; n\%2 = 1?$ is not an invariant strategy, as it at least violates **I1**: a state $\langle n = 0, m = 1 \rangle$ is a counterexample, in which no action can be executed to enforce $n\%2 = 1$.

Proposition 3. *Given a GFSS game problem $\langle \mathcal{D}, p, \phi \rangle$, an invariant strategy is a safe strategy.*

Proof. For our purpose, we use notation $W(s)$ to denote the condition $\forall s' \forall \delta'. Trans^*(\delta_{\mathcal{S}}^*, s, \delta', s') \supset \phi[s'] \wedge \exists s''. Trans(\delta_{\mathcal{S}}, s', nil, s'')$. Thus a strategy $\mathcal{S}$ is a safe strategy if $\mathcal{D} \models W(S_0)$.

We prove that if a strategy of the form $\varphi?; \pi a.a; \psi?$ is an invariant strategy, then we have $\mathcal{D} \models W(S_0)$ as follows. We first prove $\mathcal{D} \models \forall s.\varphi[s] \wedge turn(p, s) \supset W(s)$ and $\mathcal{D} \models \forall s.\psi[s] \wedge \neg turn(p, s) \supset W(s)$. Let M be a model of $\mathcal{D}$. We prove $M \models \varphi[s] \wedge turn(p, s) \wedge Trans^*(\delta_{\mathcal{S}}^*, s, \delta', s') \supset \phi[s'] \wedge \exists s''. Trans(\delta_{\mathcal{S}}, s', nil, s'')$ and $M \models \psi[s] \wedge \neg turn(p, s) \wedge Trans^*(\delta_{\mathcal{S}}^*, s, \delta', s') \supset \phi[s'] \wedge \exists s''. Trans(\delta_{\mathcal{S}}, s', nil, s'')$ for any s and s'. We prove by induction on the distance d between s and s'. Basis: $d = 0$ hence $s' = s$. We prove $M \models \varphi[s] \wedge turn(p, s) \supset \phi[s] \wedge \exists s''. Trans(\delta_{\mathcal{S}}, s, nil, s'')$,

Algorithm 1: $synthesize$

Input: GFSS game problem $\langle \mathcal{D}, p, \phi \rangle$
Output: a safe strategy or $failure$ or $unknown$

$\mathcal{P} \leftarrow predicates(\mathcal{D})$;
$\mathcal{S} \leftarrow \phi?; \pi a.a; \phi?$;
while *not timing-out* **do**
  $M \leftarrow verify(\mathcal{S}, \langle \mathcal{D}, p, \phi \rangle)$;
  if *verification succeeds* **then return** $\mathcal{S}$;
  else $\mathcal{S} \leftarrow refine(\mathcal{S}, M, \langle \mathcal{D}, p, \phi \rangle, \mathcal{P})$;
  if $\mathcal{S}$ *is* $failure$ **then return** $failure$;
return $unknown$

which follows by **I3** and **I1**; and $M \models \psi[s] \wedge \neg turn(p, s) \supset \phi[s] \wedge \exists s''.Trans(\delta_{\mathcal{S}}, s, nil, s'')$, which follows by **I3** and the requirement of FTTP games that plays are infinite sequences of states. Induction: we assume that the proposition holds when $d = n$, and proceed to prove that it holds when $d = n + 1$. So suppose that $M \models \varphi[s] \wedge turn(p, s) \wedge Trans^*(\delta^*_{\mathcal{S}}, s, \delta', s')$ and $dist(s, s') = n + 1$. Since there exists s'' and δ' such that $M \models Trans(\delta_{\mathcal{S}}, s, nil, s'') \wedge Trans^*(\delta^*_{\mathcal{S}}, s'', \delta', s')$ and $dist(s'', s') = n$, from **I1** and the axioms for $turn$, we have $M \models \psi[s''] \wedge \neg turn(p, s'') \wedge Trans^*(\delta^*_{\mathcal{S}}, s'', \delta', s')$ and $dist(s'', s') = n$. Now by induction, $M \models \phi[s'] \wedge \exists s''.Trans(\delta_{\mathcal{S}}, s', nil, s'')$. Similarly, suppose that $M \models \psi[s] \wedge \neg turn(p, s) \wedge Trans^*(\delta^*_{\mathcal{S}}, s, \delta', s')$ and $dist(s, s') = n + 1$, from **I2**, together with the axioms for $turn$ and the requirement of FTTP games, we prove $M \models \phi[s'] \wedge \exists s''.Trans(\delta_{\mathcal{S}}, s', nil, s'')$. Thus $\mathcal{D} \models \varphi[S_0] \wedge turn(p, S_0) \supset W(S_0)$ and $\mathcal{D} \models \psi[S_0] \wedge \neg turn(p, S_0) \supset W(S_0)$. By **I4** and the axioms for $turn$, it follows $\mathcal{D} \models W(S_0)$. □

We show the relations among safe postdiction strategies, invariant strategies, and first-order verifiable strategies. Here first-order verifiable strategies mean strategies that can be verified using first-order theorem proving. First, invariant strategies are a proper subset of safe postdiction strategies. Given an invariant strategy $\varphi?; \pi.a.a; \psi?$, by using a first-order formula γ denoting some unreachable states of the games, one could obtain a safe postdiction strategy $\varphi \vee \gamma?; \pi.a.a; \psi?$, which may no longer be an invariant strategy, as it may violate **I3**, as γ may induce unsafe states. Second, invariant strategies are a proper subset of first-order verifiable strategies. Given γ as mentioned above, the unreachability of γ may be proved using first-order theorem proving. Thus, $\varphi \vee \gamma?; \pi.a.a; \psi?$ can be a first-order verifiable strategy but not an invariant strategy.

In the following, we present our algorithms for the synthesis of invariant strategies.

Alg. 1 is the main procedure to synthesize an invariant strategy. We adapt the idea of counterexample-guided inductive synthesis. First, predicate symbols are extracted from the game BAT. Postdiction strategy $\mathcal{S}$ is initialized with the two formulas taken as the safety goal ϕ. While not timing out, repeat this process: verify if $\mathcal{S}$ is an invariant strategy via using an SMT solver to examine the conditions of Def. 9; if all the conditions hold, return $\mathcal{S}$; otherwise, refine $\mathcal{S}$ using the returned counterexample; if the refinement process fails, return $failure$.

Algorithm 2: $refine$

Input: strategy $\varphi?; \pi a.a; \psi?$, counterexample M, GFSS game problem $\langle \mathcal{D}, p, \phi \rangle$, predicate set $\mathcal{P}$
Output: a refined postdiction strategy or $failure$

$\mathcal{M}^+_\varphi, \mathcal{M}^-_\varphi, \mathcal{M}^+_\psi, \mathcal{M}^-_\psi \leftarrow getPosNegSet(\varphi, \psi)$;
switch M **do**
  case M *falsifies* $\varphi \models \phi$ **do** $\mathcal{M}^-_\varphi \leftarrow \mathcal{M}^-_\varphi \cup \{M\}$;
  case M *falsifies* $\psi \models \phi$ **do** $\mathcal{M}^-_\psi \leftarrow \mathcal{M}^-_\psi \cup \{M\}$;
  case M *falsifies* $\mathcal{D}_{S_0\downarrow} \models \varphi$ **do** $\mathcal{M}^+_\varphi \leftarrow \mathcal{M}^+_\varphi \cup \{M\}$;
  case M *falsifies* $\mathcal{D}_{S_0\downarrow} \models \psi$ **do** $\mathcal{M}^+_\psi \leftarrow \mathcal{M}^+_\psi \cup \{M\}$;
  case M *falsifies* **I1 do**
    $sr \leftarrow findSafeRegion(M, \langle \mathcal{D}, p, \phi \rangle)$;
    if sr *is* $failure$ **then return** $failure$;
    if sr *is* $\emptyset$ **then** $\mathcal{M}^-_\varphi \leftarrow \mathcal{M}^-_\varphi \cup \{M\}$;
    else $\mathcal{M}^+_\psi \leftarrow \mathcal{M}^+_\psi \cup succStates(sr, M)$;
  case M *falsifies* **I2 do**
    $sr \leftarrow findSafeRegion(M, \langle \mathcal{D}, p, \phi \rangle)$;
    if sr *is* $failure$ **then return** $failure$;
    if sr *is* $\emptyset$ **then** $\mathcal{M}^-_\psi \leftarrow \mathcal{M}^-_\psi \cup \{M\}$;
    else $\mathcal{M}^+_\varphi \leftarrow \mathcal{M}^+_\varphi \cup succStates(sr, M)$;
$\varphi' \leftarrow update(\varphi, \mathcal{M}^+_\varphi, \mathcal{M}^-_\varphi, k, \mathcal{P})$;
$\psi' \leftarrow update(\psi, \mathcal{M}^+_\psi, \mathcal{M}^-_\psi, k, \mathcal{P})$;
return $\varphi'?; \pi a.a; \psi'?$

Alg. 2 and Alg. 3 show the idea of counterexample-guided strategy refinement, whose intuition is as follows: a counterexample M is understood as a game state, which we call a state model, and in which the postdiction strategy fails to be an invariant strategy. As an invariant strategy represents safe strategies of all the game instances, to fix the postdiction strategy such that M no longer becomes a counterexample, we attempt to find a safe strategy of a game instance, whose safe region contains M. If we could do so, we use the information how that safe strategy behaves in the state model M, to refine the postdiction strategy. Otherwise, we let M no longer occur in the executions of the postdiction strategy.

In Alg. 2, to refine postdiction strategy $\varphi?; \pi a.a; \psi?$, we refine its formulas φ and ψ. Throughout the algorithms, we maintain $positive\ sets$ and $negative\ sets$, to indicate how to refine the formulas. A positive set denotes a set of models that should satisfy a formula, and a negative set denotes a set of models that should falsify a formula. Using $getPosNegSet$, we get the positive sets and negative sets.

We begin with deciding if counterexample M should be added to a positive set or a negative set: In case M falsifies $\varphi \models \phi$, M should be added to $\mathcal{M}^-_\varphi$, to exclude M from formula φ, thus to exclude the failure scenarios, in which the executions of the postdiction strategy lead to the state model M falsifying the safety goal ϕ. The case in which M falsifies

$\psi \models \phi$ is similar. In case M falsifies $\mathcal{D}_{S_0\downarrow} \models \varphi$, M should be added to $\mathcal{M}^+_\varphi$, as M represents an initial state of the game problem. The case in which M falsifies $\mathcal{D}_{S_0\downarrow} \models \psi$ is similar. In case M falsifies **I1**, which means that in state model M, player p cannot perform any action to enforce ψ, we do the following: we decide if state model M is $spurious$ by resorting to $findSafeRegion$ (see Alg. 3). Intuitively, we say that a state model is $spurious$ if it does not occur in any execution of the safe strategies of the game instances. If safe region sr is $\emptyset$, meaning that the state model M is spurious, we add M to $\mathcal{M}^-_\varphi$, to exclude M from formula φ, thus to exclude the occurrences of M in the executions of the postdiction strategy. Otherwise, with safe region sr, we add the successor states of M to $\mathcal{M}^+_\psi$ (using $succStates$), to guarantee that in M, following the postdiction strategy, the player can perform an action to enforce ψ. The case in which M falsifies **I2** is similar, thus we omit it here.

Example 5 cont'd. Given counterexample $\langle n = 0, m = 1\rangle$, suppose that $\mathcal{D}_{S_0}$ is $\{n(S_0) \neq m(S_0)\}$, then there is a game instance whose initial state is $\langle n = 0, m = 1\rangle$, and whose safe region is (g_1, g_2), where g_1 is $\{\langle n = 0, m = 1\rangle\}$, and g_2 is $\{\langle n = 0, m = 0\rangle\}$. As we know that the counterexample occurs in the safe region, we add $\langle n = 0, m = 0\rangle$ to $\mathcal{M}^+_\psi$.

We then update formulas φ and ψ with the new positive sets, the new negative sets, and predicate language $\mathcal{P}$. To accelerate the update process, we consider first-order formulas of the form $\phi \wedge \bigwedge\{c_1, \ldots c_n\}$, where ϕ is the safety goal, and each c_i is a clause possibly with free variables, understood as being universally quantified. A clause is the disjunction of a set of literals, where a literal is an atom or its negation. For instance, clause $\{P(x), Q(x, y)\}$ should be understood as $\forall x \forall y. P(x) \vee Q(x, y)$. With the form, given a model M satisfying the safety goal, we update a formula γ as follows: suppose we need to update γ such that $M \models \gamma$, we update each clause c_i of γ such that $M \models c_i$; suppose we need to update γ such that $M \not\models \gamma$, we update a certain clause c_j of γ such that $M \not\models c_j$. Thus, the clauses of γ share the same positive set $\mathcal{M}^+_\gamma$, while partitioning the negative set $\mathcal{M}^-_\gamma$. Formally,

- $\mathcal{M}^+_{c_i} = \mathcal{M}^+_\gamma$, for each $c_i \in \gamma$;
- $(\bigcup_{i=1}^n \mathcal{M}^-_{c_i}) = \mathcal{M}^-_\gamma$ and $\mathcal{M}^-_{c_i} \cap \mathcal{M}^-_{c_j} = \emptyset$ for $c_i, c_j \in \gamma$.

To update a formula, we follow the idea of $local\ update$ presented in [Luo and Liu 2019]. We first give the definition of $a\ change\ set$ as follows.

Definition 10. Given a clause c, a number k and two model sets $\mathcal{M}^+$ and $\mathcal{M}^-$, $change(c, k, \mathcal{M}^+, \mathcal{M}^-, \mathcal{P})$ is a clause set such that a clause c' is in it iff the following hold:

- c' is obtained from c via $changing$ at most k times, where the change operation denotes adding a literal or replacing a literal with another one;
- c' is satisfied by every model in $\mathcal{M}^+$ and is falsified by every model in $\mathcal{M}^-$.

With the above definition, to update formula γ such that $M \models \gamma$, we replace each c_i of γ with c'_i, where $c'_i \in change(c_i, k, \mathcal{M}^+_{c_i} \cup \{M\}, \mathcal{M}^-_{c_i}, \mathcal{P})$. If no such c'_i is available, we drop c_i from γ. To update γ such that $M \not\models \gamma$, we replace one c_j of γ with c'_j, where $c'_j \in \bigcup_{c_i \in \gamma} change(c_i, k, \mathcal{M}^+_{c_i}, \mathcal{M}^-_{c_i} \cup \{M\}, \mathcal{P})$. If no such c'_j is available, we restart the main algorithm if it is not timing-out. There may be multiple available clauses, for which we define a preference relation to select one. The preference relation is based on an intuition that if we choose a clause from a set of clauses, we choose that one which has less potential to cause restarts, and which has fewer literals to keep the formula simple. The definition of the preference relation is through a scoring function. Initially, the score of each literal is 0. During restart, we decrease by one the score of each literal appearing in clauses that have been chosen. The score of a clause is the sum of the scores of the literals in the clause. A clause c_1 is preferred over another clause c_2 if $score(c_1) > score(c_2)$ or $score(c_1) = score(c_2)$ and $|c_1| < |c_2|$. Note that with a clause replaced by another clause, the proof obligation might fall into an undecidable fragment, so that the SMT solvers trap into a loop. We thus set a time-out bound for the solvers, so that we are able to backtrack to select another clause for the formula update.

Algorithm 3: $findSafeRegion$

Input: counterexample M,
GFSS game problem $\langle \mathcal{D}, p, \phi\rangle$
Output: a safe region containing M or $\emptyset$ or $failure$

```
G ← grounding(D, |M| + c);
foreach G ∈ G do
    σ ← model-checking(G, ⟨⟨p⟩⟩Gφ) ;
    if σ does not exist then return failure;
    else if M ∈ G|σ then return G|σ ;
return ∅
```

Example 5 cont'd. With the new $\mathcal{M}^+_\psi$ as mentioned earlier, a possible refinement for formula ψ is to replace clause $n\%2 = 1$ with clause $n\%2 = 1 \vee m = 0$.

In Alg. 3, we attempt to find a safe region which contains M, and which is induced by a safe strategy of a certain game instance. First, from M, we attempt to generate a set of game instances. However, recall that a game BAT might possibly represent infinitely many game instances. To consider a finite subset of them, we use a bound b. The bound b is set as $|M|+c$, where c is a parameter, and $|M|$ is a number N such that $M \models B(N, s)_\downarrow$ (where B is presented in Def. 3). Intuitively, $|M|$ denotes a number such that all larger numbers are inactive throughout the state model M. Using the bound b, we consider game instances whose game active numbers are smaller than $b+1$: we first generate a finite set $\mathcal{I}$ of initial states from the initial database, that is, we generate all models of formula $\bigwedge \mathcal{D}_{S_0\downarrow} \wedge \exists k. B(k, S_0)_\downarrow \wedge k < b + 1$ via an SMT solver, and take such models as the initial states.

Example 6. In the 2-Nim game, let $\mathcal{D}_{S_0}$ be $\{n(S_0) \neq m(S_0), n(S_0) \geq 1 \wedge m(S_0) \geq 1\}$, and parameter c be 1. Suppose that a returned counterexample M is $\langle n = 1, m = 1\rangle$. Then $|M|$ can be 1, and the bound b is 2. Using the bound and an SMT solver, we get a finite set of models: $\{\langle n = 1, m = 2\rangle, \langle n = 2, m = 1\rangle\}$.

For each initial state $I \in \mathcal{I}$, we then induce a game

instance G as follows. We take bound b as the game active number, and $ground$ the action precondition axioms and successor state axioms, where the ground operation involves replacing every quantifier $\forall k$ (resp. $\exists x$) with the bounded quantifier $\forall x \leq b$ ($\exists x \leq b$). An initial state I with the grounded precondition axioms and successor state axioms represents a game instance G.

For each game instance G, we then attempt to find a safe strategy via ATL model checking. We check $\langle\langle p\rangle\rangle \mathbf{G}\phi$ over game instance G, where $\langle\langle p\rangle\rangle \mathbf{G}\phi$ expresses that agent p has a strategy to ensure that ϕ always holds. We translate a turn-based game to a concurrent game by letting a player do $null\ action$ when not in her turn. Checking $\langle\langle p\rangle\rangle \mathbf{G}\phi$ over G might fail, in which case we know that the game problem is unsolvable, as we find a game instance in which the player does not have a safe strategy. Finally, we check whether M occurs in the safe regions of the game instances and return the result.

Theorem 1. *Given a GFSS game problem, if Algorithm 1 returns a strategy, then it is a safe strategy; if $failure$ is returned, the problem is unsolvable.*

Proof. When Algorithm 1 returns a strategy, we know that it is an invariant strategy. It is also a safe strategy, which follows by Prop. 3. If $failure$ is returned, we know that there are no safe strategies for a game instance; hence there are no solutions for the GFSS game problem. □

Example 6 cont'd. Given $\mathcal{D}_{S_0}$ as before, let the safety goal be $n = 0 \wedge m = 0 \supset \neg turn(p)$. We synthesize an invariant strategy for P_1, where φ and ψ are as follows:

$$\forall x.[x < 0 \vee n \neq x \vee m \neq x] \text{ and}$$
$$\forall x.[x < 0 \vee n \neq x \vee m = x],$$

which can be further simplified to $m \neq n$ and $m = n$.

Experimentation

This section presents our implementation and the experimental results. We implemented a prototype system using Python. We use two SMT solvers for the first-order reasoning: Z3 [de Moura and Bjørner 2008] and CVC4 [Barrett et al. 2011], both of which support quantified formulas. We use the multi-agent model checker MCMAS [Lomuscio, Qu, and Raimondi 2017] for synthesizing safe strategies for FTTP games. With the SMT solvers, to check whether $\varphi \models \phi$ holds, we check whether $\varphi \wedge \neg\phi$ is unsatisfiable. We may receive the following from the SMT solvers: sat, *i.e.*, it finds a counterexample; $unsat$, *i.e.*, it proves $\varphi \models \phi$; $unknown$, *i.e.*, it is unsolvable or timing-out. To improve the success rate of the verification, when Z3 returns $unknown$, we change to CVC4. Further, we perform quantifier elimination on formulas via Z3 before sending them to the SMT solvers. The time-out bound for the main algorithm is 3600 seconds. The time-out bound for the SMT solvers is 10 seconds. Considering the computational resources, we set the number c in Alg. 3 to 1, and the number k in Alg. 2 to 3.

We did experiments with combinatorial games Nim, its variants and Chomp2xN from [Farzan and Kincaid 2018; Wu et al. 2020], and grid games ChompNxN and a variant of coloring from [Luo and Liu 2019], and a game variant of a standard protocol for leader election from [Padon et al. 2016]. If games have ending states, we adapt them into our framework by letting $null\ action$ be the only possible action when a game ends. All experiments were conducted on a MacOS machine with 2.30GHz CPU and 8GB RAM. The details of domains are as follows.

game	L	R^+	R^-	B	S	T(s)
2-Nim	44	2	5	0	6	14.5
Take-away	94	3	11	1	6	43.2
Sub.	118	7	23	0	4	321.1
E.&D.	64	3	13	0	10	210.8
Mon. 2-Nim	44	1	9	0	6	26.4
Ch.2xN	44	2	12	0	14	35.6
Ch.NxN	58	–	–	–	–	–
Coloring	32	4	11	1	8	303.2
Leader	66	0	8	2	16	367.2

Table 1: Experimental results

2-Nim: The rule is as mentioned in the Introduction.
Take-away: It is a variant of the 1-Nim. Only removing 1 or 2 or 3 pebbles is possible.
Subtraction: It is a variant of the 1-Nim. Only removing 1 or 3 or 4 pebbles is possible.
Empty-and-Divide: It is a variant of the 2-Nim. The effect of an action will remove all the pebbles from one heap and divide the pebbles of the remaining heap into two heaps.
Monotonic 2-Nim: It is a variant of the 2-Nim. A player can remove a certain number of pebbles, but can neither make any heap empty nor make the number of pebbles in the second heap more than that in the first heap.
Chomp2xN (resp.NxN): Cookies are laid out on a 2xN (resp. NxN) rectangle. The cookie in the top left position is poisoned. Two players take turns making moves: at each move, a player eats a remaining cookie, together with all cookies to the right or below it. The player who eats the poisoned cookie loses the game.
Coloring: Two players take turns to perform actions. A player paints red and another player does cleaning on a 1xN grid. Painting red on a cell is possible if the cell is not painted. Cleaning a cell is possible if the cell is painted. Show that the grid is always clean when in the player's turn.
Leader: The protocol assumes a ring of unbounded size. Every node has a unique ID. Every node can send its own ID to its neighbour in one direction. A node forwards messages containing an ID higher than its own ID. When a node receives a message with its own ID, it declares itself as a leader. Show that there is at most one leader.

Experimental results are summarized in Table 1. Here L is the number of generated literals; R^+ (resp. R^-) denotes the number of times of adding models to positive sets (resp. negative sets); B denotes the number of times of backtracking for another clause when the SMT solvers time out; S denotes the number of literals used in the strategy after simplification; T (in seconds) is the total time for synthesis. Notation **–** means timing out. Our system successfully syn-

	2-Nim	Take-away	Sub.	E.&D.	Mon. 2-Nim	Ch.2xN	Ch.NxN	Coloring	Leader
[Wu et al. 2020]	✔	✔	✔	✔	✔	✔	✘	✘	✘
Our method	✔	✔	✔	✔	✔	✔	–	✔	✔

Table 2: ✔ means that a strategy is synthesized successfully; ✘ means that the framework cannot formalize the problem.

thesizes strategies for all the domains except ChompNxN.

There are some factors affecting the running time: the number of literals, how large a game instance to solve, and how complicated the invariants to search for, *i.e.*, how many literals are in the invariants, and whether they involve quantifiers or non-arithmetic relations.

We do not directly compare with the related method presented in [Wu et al. 2020], as their problem is defined differently, in that their formalizations of domains do not include initial states but include legal states. Their method generates strategies within 250 seconds for all the tested domains, while for domain Subtraction, our method needs more than 250 seconds. However, as shown in Table 2, with the expressive framework, our method can synthesize more expressive strategies, while still in a reasonable amount of time.

Related Work

For representing and reasoning about strategies in the situation calculus, Lespérance et al. [2000] defined an action selection function $\sigma(s)$, which is a mapping from situations to primitive actions, prescribing which action the agent should perform in a situation. Xiong and Liu [2016] extended the situation calculus with a second-order strategy sort, and proposed a general strategic framework, where strategies can be compactly represented by a fragment of Golog programs. Luo and Liu [2019] proposed FSA representation of strategies and their automated verification. But none of these work concerns automated synthesis of strategies.

In the area of infinite state game solving, Farzan and Kincaid [2018] focused on linear arithmetic games, and proposed a method to solve the general problem by solving games of increasing sizes. Wu et al. [2020] proposed a faster algorithm, making use of compacted representation of states and counterexample-guided synthesis. Instead of incomplete approaches, there are also researches focusing on subclasses of game structures for which the synthesis problem is decidable, *e.g.*, push-down games [Walukiewicz 2001] and multi-counter games [Kucera 2012]. However, none of these work has enough expressiveness to formalize the games ChompNxN, Coloring and Leader and also their strategies, as these domains involve non-arithmetic relations and quantified formulas, *e.g.*, $\forall x, y.[x \neq y \supset \neg leader(x) \vee \neg leader(y)]$, expressing that there is at most one leader.

Our work is related to generalized planning. Hu and De Giacomo [2011] gave a general definition of generalized planning where generalized plans work with a set of deterministic environments. De Giacomo et al. [2016] correlated generalized planning to a game of imperfect information. These frameworks are related to the notion of two-player games, but they assume a common action pool for the planning domains. Bonet and Geffner [2018] considered relational domains where the set of actions and objects depends on the instance, by projecting actions over a common set of features. They provided an automated method to learn these features as abstractions. While the language for features is rich, involving counting and transitive closure, but without theorem-proving, their method can only guarantee approximate soundness that the plan works for the set of learning samples.

In planning, Seipp and Helmert [2013] use the idea of counterexample-guided abstraction refinement to compute abstraction heuristics for optimal classical planning. They start from a coarse abstraction of the planning task, and iteratively compute an abstract solution, by checking if and why it fails for the planning task, and refining the abstraction accordingly. The problem we focus is different, in that we focus on infinite-state systems but not finite-state systems. We refine a strategy but not a transition system; our goal of refinement is to find a solution but not an abstraction heuristic.

Conclusion

In this paper, we investigate the generalized strategy synthesis problem for a set of FTTP games with safety goals. We formalize the problem in the situation calculus. We focus on invariant strategies that aim to maintain invariants no matter how the opponent acts. We propose a sound but incomplete method to synthesize invariant strategies via counterexample-guided strategy refinement, making use of safe strategies for game instances generated with ATL model-checking. We implemented a prototype system and tested with a number of domains, and our experiments showed the viability of our approach. With the expressive framework in the situation calculus, our approach shows the potential to synthesize expressive strategies, to solve richer domains, such as those whose formalizations need quantified formulas. Although we consider two-player turn-based games, our framework is adaptable to the settings of concurrent games and multiple player games.

Our algorithm is not guaranteed to terminate, as the synthesis problem is generally undecidable. While our method is designed for broader cases, it's interesting but non-trivial to consider a class of game problems for which our algorithm will terminate. Our current approach focuses on invariant strategies, which have simple structures, while whose synthesis relies heavily on the synthesis of formulas. Introducing more complicated structures in strategies to relieve the dependence is one possible future direction.

Acknowledgments

We thank the anonymous reviewers for helpful comments. We acknowledge support from the Natural Science Foundation of China under Grant No. 62076261 and No.61902067.

References

Alur, R.; Henzinger, T. A.; and Kupferman, O. 2002. Alternating-time temporal logic. *J. ACM*, 49(5): 672–713.

Apt, K. R.; and Grädel, E. 2011. *Lectures in game theory for computer scientists*. Cambridge University Press.

Barrett, C. W.; Conway, C. L.; Deters, M.; Hadarean, L.; Jovanovic, D.; King, T.; Reynolds, A.; and Tinelli, C. 2011. CVC4. In Gopalakrishnan, G.; and Qadeer, S., eds., *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, 171–177. Springer.

Bonet, B.; and Geffner, H. 2018. Features, Projections, and Representation Change for Generalized Planning. In *IJCAI*, 4667–4673.

Bulling, N.; Dix, J.; and Jamroga, W. 2010. Model Checking Logics of Strategic Ability: Complexity. Specification and Verification of Multi-Agent Systems, M. Dastani, K. Hindriks, and J.-J. Meyer.

Chatterjee, K.; Henzinger, T. A.; and Piterman, N. 2010. Strategy logic. *Inf. Comput.*, 208(6): 677–693.

De Giacomo, G.; Murano, A.; Rubin, S.; and Stasio, A. D. 2016. Imperfect-Information Games and Generalized Planning. In *IJCAI*, 1037–1043.

de Moura, L. M.; and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In *TACAS*, 337–340.

Eén, N.; Legg, A.; Narodytska, N.; and Ryzhyk, L. 2015. SAT-Based Strategy Extraction in Reachability Games. In *AAAI*, 3738–3745.

Farzan, A.; and Kincaid, Z. 2018. Strategy synthesis for linear arithmetic games. *PACMPL*, 2(POPL): 61:1–61:30.

Genesereth, M. R.; Love, N.; and Pell, B. 2005. General Game Playing: Overview of the AAAI Competition. *AI Magazine*, 26(2): 62–72.

Giacomo, G. D.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artif. Intell.*, 121(1-2): 109–169.

Gradel, E.; Kolaitis, P. G.; Libkin, L.; Marx, M.; Spencer, J.; Vardi, M. Y.; Venema, Y.; and Weinstein, S. 2007. *Finite Model Theory and Its Applications*. Springer, Berlin.

Greenlaw, R.; Hoover, H. J.; and Ruzzo, W. L. 1995. *Limits to Parallel Computation: P-Completeness Theory*. MIT Press.

Hu, Y.; and De Giacomo, G. 2011. Generalized Planning: Synthesizing Plans that Work for Multiple Environments. In *IJCAI*, 918–923.

Kucera, A. 2012. Playing Games with Counter Automata. In *Reachability Problems - 6th International Workshop, RP 2012, Bordeaux, France, September 17-19, 2012. Proceedings*, volume 7550 of *Lecture Notes in Computer Science*, 29–41. Springer.

Lespérance, Y.; Levesque, H. J.; Lin, F.; and Scherl, R. B. 2000. Ability and Knowing How in the Situation Calculus. *Studia Logica*, 66(1): 165–186.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. *J. Log. Program.*, 31(1-3): 59–83.

Lomuscio, A.; Qu, H.; and Raimondi, F. 2017. MCMAS: an open-source model checker for the verification of multi-agent systems. *STTT*, 19(1): 9–30.

Luo, K.; and Liu, Y. 2019. Automatic Verification of FSA Strategies via Counterexample-Guided Local Search for Invariants. In *IJCAI*, 1814–1821.

McCarthy, J.; and Hayes, P. J. 1969. Some Philosophical Problems From the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4: 464–502.

Padon, O.; McMillan, K. L.; Panda, A.; Sagiv, M.; and Shoham, S. 2016. Ivy: safety verification by interactive generalization. In *PLDI, Santa Barbara, CA, USA, June 13-17*.

Pnueli, A.; and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, 179–190.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Seipp, J.; and Helmert, M. 2013. Counterexample-Guided Cartesian Abstraction Refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*. AAAI.

Solar-Lezama, A.; Tancau, L.; Bodík, R.; Seshia, S. A.; and Saraswat, V. A. 2006. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2006, San Jose, CA, USA, October 21-25, 2006*, 404–415. ACM.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *Artif. Intell.*, 175(2): 615–647.

Walukiewicz, I. 2001. Pushdown Processes: Games and Model-Checking. *Inf. Comput.*, 164(2): 234–263.

Wu, K.; Fang, L.; Xiong, L.; Lai, Z.; Qiao, Y.; Chen, K.; and Rong, F. 2020. Automatic Synthesis of Generalized Winning Strategies of Impartial Combinatorial Games Using SMT Solvers. In *IJCAI*, 1703–1711.

Xiong, L.; and Liu, Y. 2016. Strategy Representation and Reasoning in the Situation Calculus. In *ECAI*, 982–990.